[[](https://medium.com/@justin.muench?source=post_page-----ad721d4b0cde--------------------------------)](https://medium.com/@justin.muench?source=post_page-----ad721d4b0cde--------------------------------)

[Justin Muench](https://medium.com/@justin.muench?source=post_page-----ad721d4b0cde--------------------------------)

Dec 14, 2022

·

14 min read

·

Member-only

·

Listen

**.NET Core 6 — Logging and Exception Handling**



Photo by [JESHOOTS.COM](https://unsplash.com/@jeshoots?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral" \t "_blank)

Logging is a critical part of any application development process, and ASP.NET Core 6 makes it easier than ever to implement logging functionality in your web apps.

In this blog post, we'll take an in-depth look at the built-in logging features of ASP.NET Core 6 and various third-party tools you can use for enhanced logging capabilities.

Whether you're just getting started with developing in ASP.NET Core or have been coding for years, the tips included here will help optimize your production workflow from start to finish!

**Logging vs. Monitoring**

ASP.NET 6.0 provides a range of features and tools for logging and monitoring applications. At its core, logging involves capturing detailed information about the events and processes in an application, such as user actions, system events, and performance metrics. This information is typically stored in a log file or database. It can be analyzed to identify and diagnose problems, optimize performance, and gain insights into the behavior and usage of the software.

On the other hand, monitoring involves observing and measuring an application's performance and behavior in real time and providing alerts or notifications when certain conditions are met. For example, a monitoring system might be configured to send an alert if the response time of an application exceeds a certain threshold or if an error occurs. Monitoring can help identify and resolve issues before they affect users and provide valuable information for performance tuning and optimization.

Some of the popular libraries and tools for logging in ASP.NET 6.0 include Microsoft.Extensions.Logging, NLog, and Serilog.

**Why do we need logging?**

There are several reasons why logging is essential in ASP.NET 6.0 and other software development environments. Here are a few reasons:

1. Logging helps to identify and diagnose problems in the software. By capturing detailed information about the events and processes within the software, developers can use logs to identify the root causes of errors and other issues. This can reduce the time and effort required to fix bugs and improve the overall reliability of the software.
2. Logging can provide valuable insights into the performance and usage of the software. By analyzing log data, developers can better understand how the software is used, the most popular features, and where performance bottlenecks may occur. This information can be used to optimize the software and improve the user experience.
3. Logging can help to meet compliance and regulatory requirements. Some industries have legal requirements for logging specific types of information, such as user activity or financial transactions. By implementing a robust logging system, software developers can ensure that they meet these requirements and avoid potential legal penalties.
4. Logging can provide a valuable data source for machine learning and other data-driven processes. By capturing detailed information about the events and processes within the software, developers can use this data to train machine learning algorithms and improve the performance of the software over time.

**Prerequisites**

* [.NET 6.0 SDK](https://dotnet.microsoft.com/download/dotnet/6.0)
* [Visual Studio 2019](https://visualstudio.microsoft.com/downloads/) (or a compatible IDE)

**Project Structure**

I have written a sample .NET 6.0 Customer Service.

The solution includes:

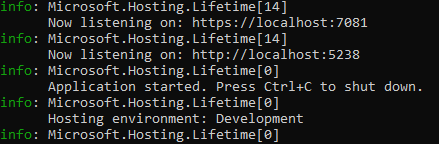
* LoggingAndMonitoringAPIExample: This is the main ASP.NET 6.0 web application, which demonstrates how to perform logging and monitoring in a web application.
* LoggingAndMonitoringAPIExample.Logic: This project contains the logic for the logging and monitoring functionality.
* LoggingAndMonitoringAPIExample.Tests: This project includes unit tests for the LoggingAndMonitoringAPIExample and LoggingAndMonitoringAPIExample.Logic projects, which test the logging and monitoring functionality.

Based on this solution, logging and monitoring will be examined and explained in more detail.

**Basic — Logging**

I want to start with the basic logging, which is included.

As soon as you start a .NET Core web application, you will notice that some info output already appears in the associated debug console. These log entries from the application tell us that the application is running and on which port and under which hosting environment it can be reached.



Generally, an ILogger<T> instance is injected into the class where the logging should occur.

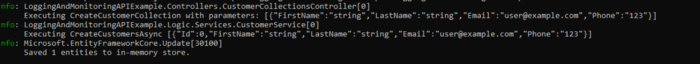
[ApiController]  
 [Route("api/[controller]")]  
 public class CustomerCollectionsController : ControllerBase   
 {  
 private readonly ICustomerService \_customerService;  
 private readonly IMapper \_mapper;  
 private readonly ILogger<CustomerCollectionsController> \_logger;  
  
  
 public CustomerCollectionsController(ICustomerService customerService, IMapper mapper, ILogger<CustomerCollectionsController> logger)  
 {  
 \_customerService = customerService;  
 \_mapper = mapper;  
 \_logger = logger;  
 }  
  
 [HttpPost]  
 [Produces("application/json", "application/xml", Type = typeof(IEnumerable<CustomerDto>))]  
 public async Task<ActionResult<IEnumerable<CustomerDto>>> CreateCustomerCollection([FromBody] IEnumerable<CustomerForCreationDto> customerCollection)  
 {  
 \_logger.LogInformation("Executing {Action} with parameters: {Parameters}", nameof(CreateCustomerCollection), JsonSerializer.Serialize(customerCollection));  
  
 var customerEntities = \_mapper.Map<IEnumerable<Customer>>(customerCollection);  
 var result = await \_customerService.CreateCustomersAsync(customerEntities);  
  
 var customerCollectionToReturn = \_mapper.Map<IEnumerable<CustomerDto>>(result);  
 var idsAsString = string.Join(",", customerCollectionToReturn.Select(x => x.Id));  
  
 return CreatedAtRoute("GetCustomers", new { customerIds = idsAsString }, customerCollectionToReturn);  
 }  
 }

It is considered best practice to create a private read-only field initialized in the constructor via DI. In addition, no more than 2–3 classes should be injected, according to Martin C. Robert.

The logger has different log methods for different logging levels; more about that later. Within the CreateCustomerCollection method, the [LogInformation](https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.loggerextensions.loginformation?view=dotnet-plat-ext-7.0" \l "microsoft-extensions-logging-loggerextensions-loginformation(microsoft-extensions-logging-ilogger-system-string-system-object())" \t "_blank) method is called. A message and the desired objects logged are passed to this method as parameters.

However, it should be noted that it is not intended to use string interpolation, even if it looks like it. The reason follows later, but the desired objects should always be passed as parameters to the method.

You should now find the log entry in the console.



Later in the post, we'll look at best practices and different locations/types for logging.

**Do's and Dont's**

* Ensure to log important events, such as when a user logs in or out, when an error occurs, or when a critical operation is performed. These events can help you troubleshoot issues and monitor the health of your application.
* Include relevant information in your logs, such as timestamps, user IDs, and details about the events logged. This can make it easier to understand and analyze your logs.
* Do use a logging framework or library to handle your application logs. This can make configuring and managing your logs easier and provide features such as log rotation and centralized storage.
* Don't log sensitive information, such as passwords or personally identifiable information (PII). These types of logs can be a security risk if unauthorized individuals access them.
* Don't use logs as a replacement for proper error handling. Your application should still have try/catch blocks and other error-handling mechanisms in place, and your logs should complement these mechanisms, not replace them.
* Store your logs in a different location than your application code. Instead, store your logs in another place, such as a centralized log server. This can make it easier to access and analyze your logs and can help prevent logs from being lost or deleted if your application code is modified or deleted.

**Log Levels and Filters**

For the logs to be helpful, you must ensure they are neat. For this purpose, filters and logging levels can be used.

**Log Levels**

**Trace**— Lowest Level and is used for low-level diagnostics to solve complex issues [1]. This should never be activated in production because it can contain sensitive data, and there will be a lot of messages [1].

**Debug** — More standard level to gather low-level diagnostic and debug data [1]. The volume of entries is also high here, so you should be careful if you apply it in a production environment [1].

**Information**—Is used for more general messages [1].

**Warning**— Is used for abnormal or unexpected events but generally without a failure in the application[1].

**Error** — While error caused a failure of the app but may be user or situation-specific [1].

**Critical** — Errors not specific to the user or situation, like out-of-disk space or crashed database [1]. The whole application is down [1].

**Log-Levels in Use**

Now let's take a look at the different log levels in use. We want to take a closer look at the Get Endpoint in the Customer Controller.

[HttpGet("{customerId}", Name = "GetCustomer")]  
 [Produces("application/json", "application/xml", Type = typeof(CustomerDto))]  
 public async Task<ActionResult<CustomerDto>> GetCustomer(int customerId)  
 {  
 \_logger.LogInformation("Executing {Action} with Id {customerId}", nameof(GetCustomer), customerId);  
   
 var customer = await \_customerService.GetCustomerAsync(customerId);  
  
 if (customer != null)  
 {  
 var result = \_mapper.Map<CustomerDto>(customer);  
  
 return Ok(result);  
 }  
  
 \_logger.LogWarning("No customer found with Id {customerId}", customerId);  
  
 return NotFound();  
 }

First, we log information on the request, including the passed CustomerId. If we do not find a customer, we log a warning message.

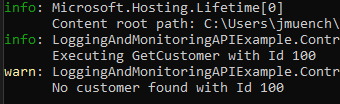
public async Task<Customer?> GetCustomerAsync(int id)  
 {  
 \_logger.LogDebug("Executing {Action} {Parameters}", nameof(GetCustomerAsync),  
 id);  
  
 var result = await \_customerContext.Customers.FirstOrDefaultAsync(x => x.Id == id);  
 return result;  
 }

In the service class, I log here in the debug level.

For example, if we have a synchronous method, we could log the load times for querying.

public Customer? GetCustomer(int id)  
{  
 var timer = new Stopwatch();  
 timer.Start();  
 var result = \_customerContext.Customers.FirstOrDefault(x => x.Id == id);  
 timer.Stop();  
  
 \_logger.LogDebug("Query took {Time} ms for {Action} and Id {Id}", timer.ElapsedMilliseconds,  
 nameof(GetCustomer), id);  
  
 return result;  
}

If we run the whole thing, we will notice that we get the warnings and information logs displayed but not the debug messages. This is because they are hidden by default.



Besides the log levels, there is also the*log category.* In the case of passing [ILogger<T>](https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.ilogger?view=dotnet-plat-ext-7.0&viewFallbackFrom=net-6.0" \t "_blank), the type corresponds to the class's namespace. Alternatively, we can select the category by injecting [ILoggerFactory](https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.iloggerfactory?view=dotnet-plat-ext-7.0&viewFallbackFrom=net-6.0" \t "_blank) instead of ILogger<T>.

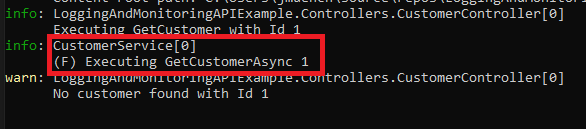
private readonly ILogger \_factoryLogger;  
private readonly ILogger<CustomerService> \_logger;  
  
public CustomerService(CustomerDbContext customerContext, ILoggerFactory loggerFactory, ILogger<CustomerService> logger)  
{  
 \_customerContext = customerContext;  
   
 \_factoryLogger = loggerFactory.CreateLogger("CustomerServiceCategory");  
   
 \_logger = logger;  
}

For demonstration purposes, I have now injected both the ILogger and the IFactoryLogger.

I updated the GetCustomerAsync Method and added the \_factoryLogger as well.

public async Task<Customer?> GetCustomerAsync(int id)  
{  
 \_logger.LogDebug("Executing {Action} {Parameters}", nameof(GetCustomerAsync),  
 id);  
   
 \_factoryLogger.LogInformation("(F) Executing {Action} {Parameters}", nameof(GetCustomerAsync),  
 id);  
  
 var result = await \_customerContext.Customers.FirstOrDefaultAsync(x => x.Id == id);  
 return result;  
}

As you can see below, the log category is shown as well.



**Log Filters**

Log filters help us to control what is written into our logs [1].

They are set up in three parts [1]:

Provider + Category + Minimum Level [1]

The Provider is where are logs are written. The category is shown in the example above, and the minimum level for the combination [1].

Log filters are the essential mechanism for controlling noise in the logs [1]. In high-traffic environments, it is necessary to be careful because they can produce a lot of messages [1].

It is best practice to specify the filters in the configuration (appsettings, environment variables, command line, etc.), but it is also possible to set it in the code [1]. The benefit of using the configuration is that you can change it during the run time [1].

Let's first show the missing debug log messages. For this, we open appsettings.json in the root directory.

{  
 "Logging": {  
 "LogLevel": {  
 "Default": "Information",  
 "Microsoft.AspNetCore": "Warning"  
 }  
 },  
 "AllowedHosts": "\*"  
}

It sets the minimum log level to information. This is why we do not see any debug entries [1].

We should also see the debug entries if we change the default level to debug and rerun our code.

https://miro.medium.com/max/631/1*KsxDfMz9KmdZAA1M13wGmA.png

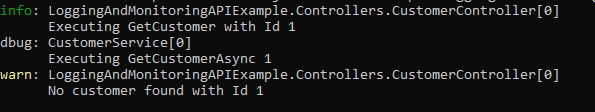
*(If you don't see them, pick the right appsettings for the environment. If the running environment is set up to development in the launch settings, then you have to change the appsettings.Development.json)*

If we want to set the LogLevel differently for a category, we can list the specific category in the appsettings.

{  
 "Logging": {  
 "LogLevel": {  
 "Default": "Information",  
 "CustomerService": "Debug",  
 "Microsoft.AspNetCore": "Warning"  
 }  
 }  
}

public CustomerService(CustomerDbContext customerContext, ILoggerFactory loggerFactory)  
{  
 \_customerContext = customerContext;  
 \_logger = loggerFactory.CreateLogger("CustomerService");  
}

In this case, we only see the debug entries from our CustomerService logger, not all other debug entries.



Alternatively, the log level can be set in the environment variables in the launchsettings.

"profiles": {  
"LoggingAndMonitoringAPIExample": {  
 "commandName": "Project",  
 "dotnetRunMessages": true,  
 "launchBrowser": true,  
 "launchUrl": "swagger",  
 "applicationUrl": "https://localhost:7081;http://localhost:5238",  
 "environmentVariables": {  
 "ASPNETCORE\_ENVIRONMENT": "Development",  
 //Specify log level... below  
 "Logging\_\_LogLevel\_\_CustomerService": "Debug"   
 }  
}

**Providers**

Next, let's look at the providers. These are the destination where the log entries are written [1].

Let's assume we want two different providers and logging different levels.

In this case, we can wrap the Log Level in the app settings with the console like below:

{  
 "Logging": {  
 "Console": {  
 "LogLevel": {  
 "Default": "Information",  
 "CustomerService": "Debug",  
 "Microsoft.AspNetCore": "Warning"  
 }  
 }  
 }  
}

A second provider can be Debug, using the system.Diagnostics.Debug, which writes to TraceListeners [1].

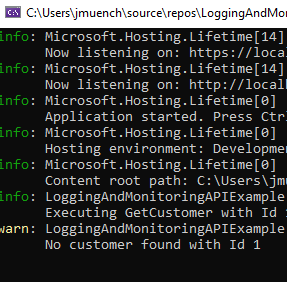
{   
 "Logging": {  
 "Console": {  
 "LogLevel": {  
 "Default": "Information",  
 "CustomerService": "Information",  
 "Microsoft.AspNetCore": "Warning"  
 }  
 },  
 "Debug": {  
 "LogLevel": {  
 "Default": "Information",  
 "CustomerService": "Debug",  
 "Microsoft.AspNetCore": "Warning"  
 }  
 }  
 }  
}

We added a second section in our appsettings wrapped in "Debug."

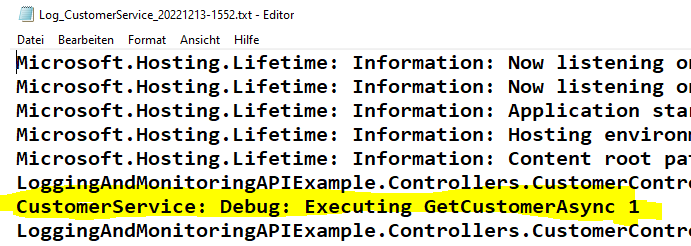
To see the debug messages a little bit clearer, we added in our Programm.cs [1]:

var path = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);  
var tracePath = Path.Join(path, $"Log\_CustomerService\_{DateTime.Now.ToString("yyyyMMdd-HHmm")}.txt");  
Trace.Listeners.Add(new TextWriterTraceListener(tracePath));  
Trace.AutoFlush = true;

Now we can call the GetCustomer Method and look at the file and console.



The file should be stored in your local AppData directory.



As you can see, we do log debug entries into the file, but they are not shown in the console.

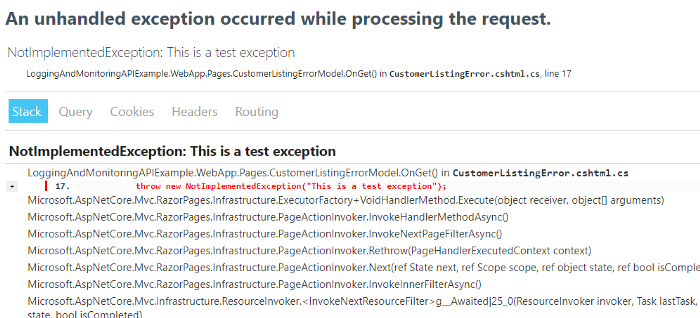
**Exception Handling and Logging**

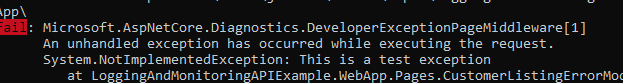
To demonstrate error logging, I added a WebApp to the project that will eventually call the CustomerCollections.

In addition, I also added an endpoint in the API for throwing an exception and another on a new page like the following:

private readonly HttpClient \_apiClient;  
 public CustomerListingErrorModel(HttpClient apiClient)  
 {  
 \_apiClient = apiClient;  
 \_apiClient.BaseAddress = new Uri("https://localhost:7081/");  
 }  
  
 public void OnGet()  
 {  
 throw new NotImplementedException("This is a test exception");  
 }

That page throws an error when called. However, this is not an excellent way — to display this error, as seen in the picture below.





This is a very trivial example, but distributed system errors are common, not necessarily because of application errors but because of the many components that can go wrong. To find these errors, decent error handling must be built in.

**Principles**

Exception handling is a critical principle to consider when developing an application [1]. The primary goal should be to always provide an elegant user experience by shielding users from errors and preventing them from seeing details that could compromise the security of their data or the system [1].

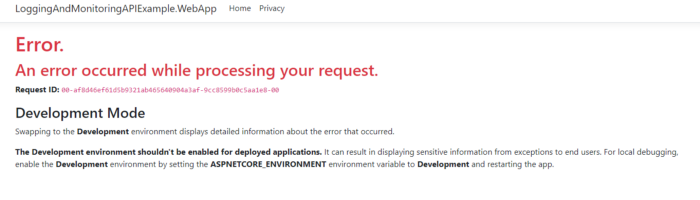
Applying global exception handling within the application architecture helps to achieve this, as well as leveraging try/catch blocks when specific errors need to be addressed individually [1].

Additionally, receiving and properly managing generated error IDs for easy referencing at a later time is imperative [1].

Meanwhile, knowing how to rely on logs during local development can help ensure that any errors are caught and logged before they can cause further harm down the line [1].

Try/catch blocks should only be used when adding value with them [1]. For example, if you need to add additional data to an exception when you make a Linq SQL request or so [1]. Alternatively, you can use try/catch if you want to continue processing in an exceptional case [1].

I have now enabled redirection to the error page in the web app for the Dev environment, so we get a better error page than the one above.



https://miro.medium.com/max/604/1*KCm0xLlF_FzhavVq4ZDZ4w.png

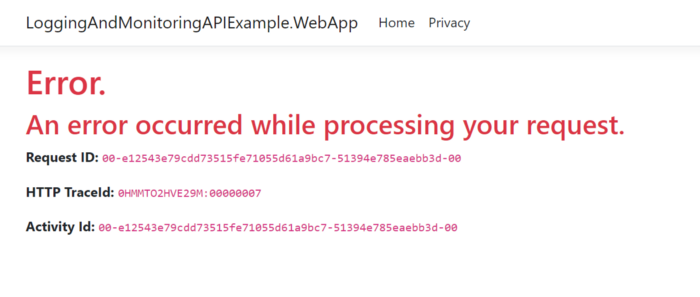
But we want a little more information and do not need the development mode message.

I added the Activity and TraceId to the ErrorModel.

public class ErrorModel : PageModel  
{  
 public string? RequestId { get; set; }  
  
 public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);  
  
 public string TraceId { get; set; }  
  
 public Activity? CurrentActiviy { get; set; }  
  
 private readonly ILogger<ErrorModel> \_logger;  
  
 public ErrorModel(ILogger<ErrorModel> logger)  
 {  
 \_logger = logger;  
 }  
  
 public void OnGet()  
 {  
 RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier;  
 TraceId = HttpContext.TraceIdentifier ?? string.Empty;  
 CurrentActiviy = Activity.Current;  
 }

Furthermore, I made some changes to the Error Page because most of the information shown on the error page is irrelevant to the user.

@page  
@model ErrorModel  
@{  
 ViewData["Title"] = "Error";  
}  
  
<h1 class="text-danger">Error.</h1>  
<h2 class="text-danger">An error occurred while processing your request.</h2>  
  
@if (Model.ShowRequestId)  
{  
 <p>  
 <strong>Request ID:</strong> <code>@Model.RequestId</code>  
 </p>  
 <p>  
 <strong>HTTP TraceId:</strong> <code>@Model.TraceId</code>  
 </p>  
 <p>  
 <strong>Activity Id:</strong> <code>@Model.CurrentActiviy?.Id</code>  
 </p>  
}

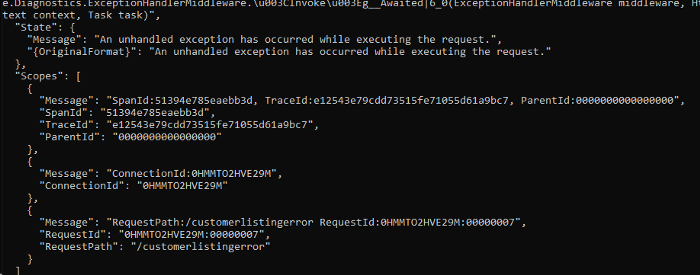


But we, as developers, need information like Ids to find the error on the logs.

The exception shown in the console is hard to read, so we added some formatting to the [appsettings](https://learn.microsoft.com/en-us/dotnet/core/extensions/console-log-formatter" \l "set-formatter-with-configuration" \t "_blank), making it much easier to see what happened.

{  
 "Logging": {  
 "LogLevel": {  
 "Default": "Information",  
 "Microsoft.AspNetCore": "Warning"  
 },  
 "Console": {  
 "FormatterName": "json",  
 "FormatterOptions": {  
 "SingleLine": true,  
 "IncludeScopes": true,  
 "TimestampFormat": "HH:mm:ss ",  
 "UseUtcTimestamp": true,  
 "JsonWriterOptions": {  
 "Indented": true  
 }  
 }  
 }  
 },  
 "AllowedHosts": "\*"  
}

And as you can see in the console, the error will be formatted.



So we looked at how to make the bugs look more likable in the UI. However, there is no error page when we call an API.

But again, we only want to provide a little information but ensure that the client gets the received errors in a format it can handle. For this, there is a [convention](https://www.rfc-editor.org/rfc/rfc7807). In .NET 7.0, there is already a [middleware](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/error-handling?view=aspnetcore-7.0#pds7) for this.

Unfortunately, in .NET 6.0, not. But there is a NuGet Package called [Hellang.Middleware.ProblemDetails](https://www.nuget.org/packages/Hellang.Middleware.ProblemDetails/" \t "_blank), which will help us [1].

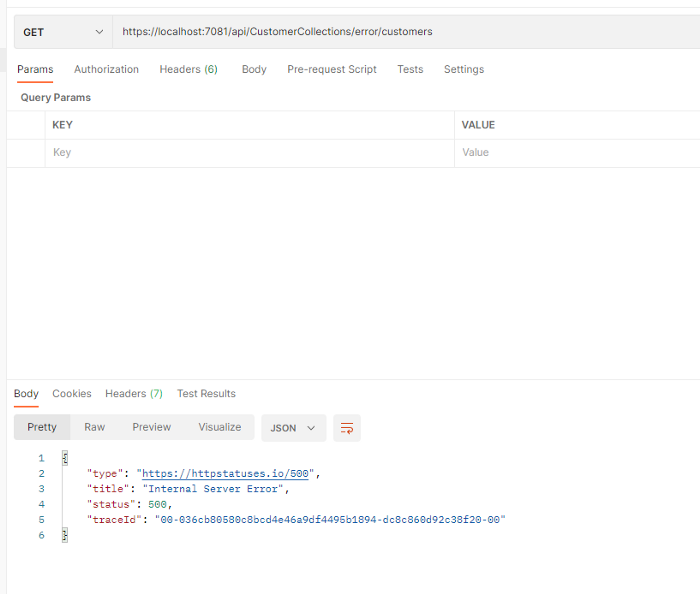
Adding this to the API Project:

builder.Services.AddProblemDetails();

and

app.UseProblemDetails();

Let's change to the production environment and rerun the API endpoint.



As you can see, the error message is now formatted in a way the client excepts it because of RFC7807. But there is one more thing we should change. The response can contain exception details by default [1].

To change this, we can modify the AddProblemDetails Method in the program.cs [1].

builder.Services.AddProblemDetails(opts =>  
{  
 opts.IncludeExceptionDetails = (ctx, ex) => false;  
 opts.OnBeforeWriteDetails = (ctx, dtls) =>  
 {  
 if (dtls.Status == 500)  
 dtls.Detail = "An error occured. Use Trace Id when contacting us.";  
 };  
});

Now we get uniform error responses, which is why we can customize the WebApp to handle the errors properly.

I injected a logger into my CustomerListingModel class.

public class CustomerListingModel : PageModel  
{  
 private readonly HttpClient \_apiClient;  
 private readonly ILogger<CustomerListingModel> \_logger;  
 public IEnumerable<CustomerDto> Customers { get; set; }  
 public CustomerListingModel(HttpClient apiClient, ILoggerFactory loggerFactory)  
 {  
 \_apiClient = apiClient;  
 \_apiClient.BaseAddress = new Uri("https://localhost:7081/");  
 \_logger = loggerFactory.CreateLogger<CustomerListingModel>();  
 }  
  
 public async Task OnGetAsync()  
 {  
 var response = await \_apiClient.GetAsync("api/CustomerCollections/error/customers");  
  
 if (!response.IsSuccessStatusCode)  
 {  
 var details = await response.Content.ReadFromJsonAsync<ProblemDetails>() ?? new ProblemDetails();  
  
 \_logger.LogWarning("API failure: {adress} Response {response}, Trace {traceId}",  
 $"{\_apiClient.BaseAddress}api/CustomerCollections", response.StatusCode, details.Extensions["traceId"]?.ToString());  
 throw new Exception("Api call failed");  
 }  
  
 Customers = await response.Content.ReadFromJsonAsync<List<CustomerDto>>() ?? new List<CustomerDto>();  
 }  
}

Through this logging and the unified response via ProblemDetails, we can deserialize the response and get the TraceId. We can log this away and show it to the user at the same time.

This is more text than I expected, so I will probably write a second part here.

**Conclusion**

In conclusion, logging is essential to any application built with ASP.NET Core 6 or other modern technologies like NodeJS or JavaEE 8+.

Correctly implemented logs can provide invaluable insights into user behavior and usage patterns and serve as early warning signs for potential issues that could arise later on down the line — making them valuable tools in any developer's toolbox!

With this guide in hand, you should now have a better understanding of how to get started with logging in to ASP.NET Core 6 so that you can begin taking advantage of this powerful feature today!

Want to connect?  
You can contact me via my social media channels ([Twitter](https://twitter.com/muench_justin), [LinkedIn](https://www.linkedin.com/in/justin-m%C3%BCnch-0b1087133/)).

**References**

1. [Logging and Monitoring in ASP.NET Core 6 by Erik Dahl](https://app.pluralsight.com/library/courses/logging-monitoring-aspdotnet-core-6/table-of-contents)
2. <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/logging/?view=aspnetcore-6.0>

**Repo**

**[GitHub - muench-develops/LoggingAndMonitoringAPIExample: Test Project API Design](https://github.com/muench-develops/LoggingAndMonitoringAPIExample" \t "_blank)**

[To get started with the project, you will need to have the following prerequisites installed: Once you have the…](https://github.com/muench-develops/LoggingAndMonitoringAPIExample" \t "_blank)

[github.com](https://github.com/muench-develops/LoggingAndMonitoringAPIExample" \t "_blank)

[Aspnetcore](https://medium.com/tag/aspnetcore?source=post_page-----ad721d4b0cde---------------aspnetcore-----------------)

[Programming](https://medium.com/tag/programming?source=post_page-----ad721d4b0cde---------------programming-----------------)

[Engineering](https://medium.com/tag/engineering?source=post_page-----ad721d4b0cde---------------engineering-----------------)

[Logging](https://medium.com/tag/logging?source=post_page-----ad721d4b0cde---------------logging-----------------)

[Exception Handling](https://medium.com/tag/exception-handling?source=post_page-----ad721d4b0cde---------------exception_handling-----------------)

62

Enjoy the read? Reward the writer.Beta

Your tip will go to Justin Muench through a third-party platform of their choice, letting them know you appreciate their story.

Give a tip

Keep up with the latest in AI, tech, and .NET development

Subscribe to my email list to receive the latest news, tips, and insights on AI, tech, and .NET development. From code correction to unit testing, I'll inform you of the latest trends and best practices in these cutting-edge fields. Sign up now to stay ahead of the curve.

Emails will be sent to herbertvaughn57@gmail.com. [Not you?](https://medium.com/m/signin?operation=login&redirect=https%3A%2F%2Fmedium.com%2Fcodex%2Fasp-net-core-6-logging-ad721d4b0cde&newsletterV3=feb470108eb8&newsletterV3Id=6c36af9215fb&source=--------------------------subscribe_user----------6c36af9215fb-)

Subscribe

[More from CodeX](https://medium.com/codex?source=post_page-----ad721d4b0cde--------------------------------)

Following

Everything connected with Tech & Code. Follow to join our 1M+ monthly readers

[[Brian Olson](https://medium.com/@devblabs?source=post_page-----ad721d4b0cde----0----------------------------)](https://medium.com/@devblabs?source=post_page-----ad721d4b0cde----0----------------------------)

[Brian Olson](https://medium.com/@devblabs?source=post_page-----ad721d4b0cde----0----------------------------)

[·Dec 14, 2022](https://medium.com/codex/your-first-question-when-writing-a-doc-bd3388bb8b49?source=post_page-----ad721d4b0cde----0----------------------------)

Member-only

**[Your First Question When Writing a Doc](https://medium.com/codex/your-first-question-when-writing-a-doc-bd3388bb8b49?source=post_page-----ad721d4b0cde----0----------------------------)**

[Disclaimer: All opinions are my own As a software engineer, I do a surprising amount of writing Design docs Blogs like this one (ha!) Feedback for performance reviews Documentation for products I work on Explanations of problems and solutions Chat messages for individuals and my team Emails (as little as…](https://medium.com/codex/your-first-question-when-writing-a-doc-bd3388bb8b49?source=post_page-----ad721d4b0cde----0----------------------------)

[Engineering](https://medium.com/tag/engineering?source=post_page-----ad721d4b0cde---------------engineering-----------------)

[3 min read](https://medium.com/codex/your-first-question-when-writing-a-doc-bd3388bb8b49?source=post_page-----ad721d4b0cde----0----------------------------)

[[](https://medium.com/codex/your-first-question-when-writing-a-doc-bd3388bb8b49?source=post_page-----ad721d4b0cde----0----------------------------)](https://medium.com/codex/your-first-question-when-writing-a-doc-bd3388bb8b49?source=post_page-----ad721d4b0cde----0----------------------------)

Share your ideas with millions of readers.

[Write on Medium](https://medium.com/new-story?source=post_page_footer_cta_write-------------------------------------)

[[Jeffrey Clos](https://medium.com/@jclos2679?source=post_page-----ad721d4b0cde----1----------------------------)](https://medium.com/@jclos2679?source=post_page-----ad721d4b0cde----1----------------------------)

[Jeffrey Clos](https://medium.com/@jclos2679?source=post_page-----ad721d4b0cde----1----------------------------)

[·Dec 14, 2022](https://medium.com/codex/the-geneinno-s2-underwater-scooter-making-light-of-ocean-lake-exploration-dd4c75cadd9?source=post_page-----ad721d4b0cde----1----------------------------)

Member-only

**[The Geneinno S2 Underwater Scooter, Making Light of Ocean/Lake Exploration](https://medium.com/codex/the-geneinno-s2-underwater-scooter-making-light-of-ocean-lake-exploration-dd4c75cadd9?source=post_page-----ad721d4b0cde----1----------------------------)**

[Small, Powerful, and Most of all, Fun! — Jeffrey Clos is a participant in the Amazon Associates LLC associates program, an affiliate advertising program designed to provide a means for sites to earn advertising fees by advertising and linking to Amazon.com. There are some links within this article to the described product. If you’re looking to explore the…](https://medium.com/codex/the-geneinno-s2-underwater-scooter-making-light-of-ocean-lake-exploration-dd4c75cadd9?source=post_page-----ad721d4b0cde----1----------------------------)

[Electric Vehicles](https://medium.com/tag/electric-vehicles?source=post_page-----ad721d4b0cde---------------electric_vehicles-----------------)

[3 min read](https://medium.com/codex/the-geneinno-s2-underwater-scooter-making-light-of-ocean-lake-exploration-dd4c75cadd9?source=post_page-----ad721d4b0cde----1----------------------------)

[[](https://medium.com/codex/the-geneinno-s2-underwater-scooter-making-light-of-ocean-lake-exploration-dd4c75cadd9?source=post_page-----ad721d4b0cde----1----------------------------)](https://medium.com/codex/the-geneinno-s2-underwater-scooter-making-light-of-ocean-lake-exploration-dd4c75cadd9?source=post_page-----ad721d4b0cde----1----------------------------)

[[Techstack](https://medium.com/@techstack-ltd?source=post_page-----ad721d4b0cde----2----------------------------)](https://medium.com/@techstack-ltd?source=post_page-----ad721d4b0cde----2----------------------------)

[Techstack](https://medium.com/@techstack-ltd?source=post_page-----ad721d4b0cde----2----------------------------)

[·Dec 14, 2022](https://medium.com/codex/how-to-craft-a-test-strategy-document-sample-test-strategy-inside-cffbdd291491?source=post_page-----ad721d4b0cde----2----------------------------)

**[How To Craft a Test Strategy Document [Sample Test Strategy Inside]](https://medium.com/codex/how-to-craft-a-test-strategy-document-sample-test-strategy-inside-cffbdd291491?source=post_page-----ad721d4b0cde----2----------------------------)**

[Without testing documentation, measuring QA quality remains challenging even for seasoned professionals. The need for more documentation is even more apparent when onboarding new QA engineers and scaling your product. With documentation, the value and purpose of quality assurance remain clear. When quality is measured correctly, you can predict test…](https://medium.com/codex/how-to-craft-a-test-strategy-document-sample-test-strategy-inside-cffbdd291491?source=post_page-----ad721d4b0cde----2----------------------------)

[Quality Assurance](https://medium.com/tag/quality-assurance?source=post_page-----ad721d4b0cde---------------quality_assurance-----------------)

[9 min read](https://medium.com/codex/how-to-craft-a-test-strategy-document-sample-test-strategy-inside-cffbdd291491?source=post_page-----ad721d4b0cde----2----------------------------)

[[](https://medium.com/codex/how-to-craft-a-test-strategy-document-sample-test-strategy-inside-cffbdd291491?source=post_page-----ad721d4b0cde----2----------------------------)](https://medium.com/codex/how-to-craft-a-test-strategy-document-sample-test-strategy-inside-cffbdd291491?source=post_page-----ad721d4b0cde----2----------------------------)

[[Lucky Brian](https://medium.com/@luckybriann?source=post_page-----ad721d4b0cde----3----------------------------)](https://medium.com/@luckybriann?source=post_page-----ad721d4b0cde----3----------------------------)

[Lucky Brian](https://medium.com/@luckybriann?source=post_page-----ad721d4b0cde----3----------------------------)

[·Dec 14, 2022](https://medium.com/codex/metaverse-development-company-in-usa-9b816204c1d8?source=post_page-----ad721d4b0cde----3----------------------------)

**[Insights On The Metaverse Technology And Its Future Potential](https://medium.com/codex/metaverse-development-company-in-usa-9b816204c1d8?source=post_page-----ad721d4b0cde----3----------------------------)**

[Insights On The Metaverse Technology And Its Future Potential — The Metaverse is a post-reality universe that combines the real world with digital virtual worlds. It is a place where many people can interact with each other all the time. The Metaverse is built on the merging of augmented reality (AR) and virtual reality (VR) technologies, which make it possible…](https://medium.com/codex/metaverse-development-company-in-usa-9b816204c1d8?source=post_page-----ad721d4b0cde----3----------------------------)

[Metaverse Development](https://medium.com/tag/metaverse-development?source=post_page-----ad721d4b0cde---------------metaverse_development-----------------)

[4 min read](https://medium.com/codex/metaverse-development-company-in-usa-9b816204c1d8?source=post_page-----ad721d4b0cde----3----------------------------)

[[](https://medium.com/codex/metaverse-development-company-in-usa-9b816204c1d8?source=post_page-----ad721d4b0cde----3----------------------------)](https://medium.com/codex/metaverse-development-company-in-usa-9b816204c1d8?source=post_page-----ad721d4b0cde----3----------------------------)

[[Dmitry Yarygin](https://medium.com/@nomadic-dmitry?source=post_page-----ad721d4b0cde----4----------------------------)](https://medium.com/@nomadic-dmitry?source=post_page-----ad721d4b0cde----4----------------------------)

[Dmitry Yarygin](https://medium.com/@nomadic-dmitry?source=post_page-----ad721d4b0cde----4----------------------------)

[·Dec 13, 2022](https://medium.com/codex/test-automation-in-your-browser-no-local-setup-needed-11bcbd64765c?source=post_page-----ad721d4b0cde----4----------------------------)

Member-only

**[Test Automation in your Browser. No local setup needed?](https://medium.com/codex/test-automation-in-your-browser-no-local-setup-needed-11bcbd64765c?source=post_page-----ad721d4b0cde----4----------------------------)**

[Test Automation is one of the essential tasks of the QA Engineer now. But most of our current work involves preparing the environment and setting up tests complexly. I was researching my options for running tests right in my browser without too much configuration effort. I immediately got the idea…](https://medium.com/codex/test-automation-in-your-browser-no-local-setup-needed-11bcbd64765c?source=post_page-----ad721d4b0cde----4----------------------------)

[Programming](https://medium.com/tag/programming?source=post_page-----ad721d4b0cde---------------programming-----------------)

[4 min read](https://medium.com/codex/test-automation-in-your-browser-no-local-setup-needed-11bcbd64765c?source=post_page-----ad721d4b0cde----4----------------------------)