

Logging in Real-World Applications

How do you ensure your application runs **successfully without errors**? And if errors occur, how do you **track them**? You need to **record all actions and errors** that happen during application execution. This process is called **logging**. Whenever execution reaches a **specific method** (e.g., a controller action method), a **log message** is recorded.

- This message is **stored** for later review.
- If a **user reports an error**, you can check the logs to get **error details**.
- Logs serve as a **record** of all actions and errors that occur during execution.

This helps in **debugging, monitoring, and maintaining** application performance. ☺

Understanding Logging in ASP.NET Core

When a request reaches the controller, a log message is recorded using a logging framework.

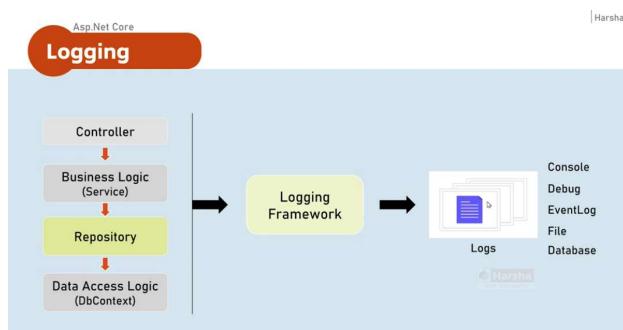
- The controller then calls the service, and another log message is recorded.
- The same process happens at **each layer** of the application.

Logging Frameworks & Logs

- The logging framework acts as a **mediator** between your **application layers** and the **log storage**.
- Logs are data stores where log messages are **persisted** for later review.
- Console logs are temporary and immediately displayed in the Kestrel window.
- Other logs can be stored for **long-term tracking and analysis**.

Using Serilog for Logging

- Create and manage logs in ASP.NET Core.
- Use Serilog, a popular third-party logging framework.
- Understand why logging is crucial for tracking application health and why it's a common interview topic.





Asp .Net Core | Harsha

ILogger

Debug ` ILogger.LogDebug("log_message");`	Logs that provide details & values of variables for debugging purpose.
Information ` ILogger.LogInformation("log_message");`	Logs that track the general flow of the application execution.
Warning ` ILogger.LogWarning("log_message");`	Logs that highlight an abnormal or unexpected event.
Error ` ILogger.LogError("log_message");`	Logs to indicate that flow of execution is stopped due to a failure.
Critical ` ILogger.LogCritical("log_message");`	Logs to indicate an unrecoverable application crash.

Understanding Log Levels in ASP.NET Core

Logging levels categorize logs based on their importance and severity. The most commonly used log levels are:

- Debug - Used for debugging information, like tracking variable values.
- Information - Logs general flow of execution (e.g., when a request reaches a controller).
- Warning - Indicates unexpected but non-critical behavior (e.g., a condition not met as expected).
- Error - Logs exceptions and issues that need attention.
- Critical - Logs severe issues like application crashes or critical failures.

Note: There is also a **Trace** level, but it's rarely used in real-world projects.

Examples of Log Levels in Action

Debug Log Example
Tracking a loop execution:

```
for (int i = 0; i < 10; i++)
{
    _logger.LogDebug($"Current loop iteration: {i}");
}
```

Information Log Example

Logging when a request reaches a controller:

```
_logger.LogInformation("GET request received at /api/users");
```

Warning Log Example

Unexpected condition in a boolean check:

```
if (isDeleted)
{
    _logger.LogInformation("Record is marked as deleted.");
}
```

Error Log Example

Logging an exception:

```
try
{
    // Database operation
}
catch (Exception ex)
{
    _logger.LogError(ex, "Error occurred while updating the database.");
}
```

Critical Log Example

Logging a server crash scenario:

```
_logger.LogCritical("Application encountered a fatal error and is shutting down.");
```

Why Logging Matters

- Debugging made easy – Quickly track and diagnose issues.
- Better maintainability – Logs help in identifying unexpected behavior.
- Monitoring & Auditing – Helps track system performance and failures.

Next, let's see how to implement these logs in ASP.NET Core! ↗

In Program.cs file

```
27 //create application pipeline
28 if (builder.Environment.IsDevelopment())
29 {
30     app.UseDeveloperExceptionPage();
31 }
32
33
34 app.Logger.LogDebug("debug-message");
35 app.Logger.LogInformation("information-message");
36 app.Logger.LogWarning("warning-message");
37 app.Logger.LogError("error-message");
38 app.Logger.LogCritical("critical-message");
39
40 if (builder.Environment.IsEnvironment("Test") == false)
41     Rotativa.AspNetCore.RotativaConfiguration.Setup("wwwroot",
42         "wkhtmltopdfRelativePath: \"Rotativa\"");
43
44 app.UseStaticFiles();
```

Run and see the output in the kestrel window.

Output in Kestrel Window:

```
[1] D:\ASSIGNMENT\Practice\ASP.NET CORE\Asp .Net Core\9-.NET-Ultimate-Guide\Section_26Logging and Serilog\CRUDExample
info: CRUDExample[0]
Information: 
warn: CRUDExample[0]
warning: 
crit: CRUDExample[0]
error: 
fatal: CRUDExample[0]
critical: Microsoft.Hosting.Lifetime[0]
info: Microsoft.Hosting.Lifetime[0]
Microsoft.Hosting.Lifetime[0]: Now listening on: http://localhost:5195
info: Microsoft.Hosting.Lifetime[0]
Microsoft.Hosting.Lifetime[0]: Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
Content root path: D:\ASSIGNMENT\Practice\ASP.NET CORE\Asp .Net Core\9-.NET-Ultimate-Guide\Section_26Logging and Serilog\CRUDExample
info: Microsoft.EntityFrameworkCore.Database.Command[20989]
Selected database 'CRMDB' (Default) (ConnectionStringName='Text', CommandTimeout=30)
SELECT [p].[PersonID] AS [p], [Address] AS [p], [CountryID] AS [p], [DateOfBirth], [p].[Email], [p].[Gender], [p].[PersonName], [p].[ReceiveNewsLetters], [p].[TaxIdentificationNumber], [c].[CountryID], [c].[CountryName]
yName
FROM [Person] AS [p]
LEFT JOIN [Countries] AS [c] ON [p].[CountryID] = [c].[CountryID]
```

But where is our 'debug log'? Because the default log level is asp.net core application is set to **information**. So all the log levels starting from information up to critical, all these are logged except debug and trace.

By using the configuration files that is appsettings.json, you can change the log settings.

You can set the minimum log level to Debug or Trace.

Of course, Trace is not much used in real-world projects. At least you can set it to Debug.

But how do you do it practically?

We will learn to set the configuration settings in the next lecture.



Got it! Here's a cleaned-up and more concise version of your text with better readability:

Did you notice a problem?

We're writing Debug logs, but they're not appearing. That's because, by default, only logs with a level of Information or higher are recorded. The default log level is Information.

Understanding Log Levels

ASP.NET Core supports the following log levels, ranked by severity:

- Trace (rarely used in real projects)
- Debug (for tracking variable values and debugging)
- Information (for general execution flow, e.g., tracking method calls)
- Warning (for unexpected but non-breaking issues, like unexpected conditions)
- Error (for exceptions and runtime errors)
- Critical (for severe failures like application crashes)

By default, logs below Information (such as Debug and Trace) are ignored.

Changing the Minimum Log Level

You can modify the minimum

Did you notice a problem?

We're writing Debug logs, but they're not appearing. That's because, by default, only logs with a level of Information or higher are recorded. The default log level is Information.

Understanding Log Levels

ASP.NET Core supports the following log levels, ranked by severity:

- Trace (rarely used in real projects)
- Debug (for tracking variable values and debugging)
- Information (for general execution flow, e.g., tracking method calls)
- Warning (for unexpected but non-breaking issues, like unexpected conditions)
- Error (for exceptions and runtime errors)
- Critical (for severe failures like application crashes)

By default, logs below Information (such as Debug and Trace) are ignored.

Changing the Minimum Log Level

You can modify the minimum log level in appsettings.json. However, keep in mind that ASP.NET Core also loads environment-specific configuration files, such as appsettings.Development.json. Since it loads after appsettings.json, it overrides any conflicting settings.

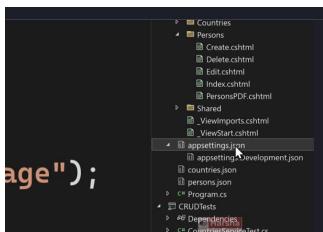
To ensure your changes take effect, update appsettings.Development.json instead. Locate the Logging section and adjust the LogLevel settings to capture lower-severity logs like Debug.

log level in appsettings.json. However, keep in mind that ASP.NET Core also loads environment-specific configuration files, such as appsettings.Development.json. Since it loads after appsettings.json, it overrides any conflicting settings.

To ensure your changes take effect, update appsettings.Development.json instead. Locate the Logging section and adjust the LogLevel settings to capture lower-severity logs like Debug.

The screenshot shows a code editor with the file 'appsettings.json' open. The JSON content is as follows:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug | Information | Warning | Error | Critical",
      "Microsoft.AspNetCore": "Debug | Information | Warning | Error | Critical"
    }
  }
}
```



The screenshot shows a code editor with the 'appsettings.json' file open. The 'Logging' section has been modified to include 'Information' and 'Warning' levels:

```

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

```

    info: CRUDExample[0]
    info: CRUDExample[0]
    information-message
    warn: CRUDExample[0]
    warning-message
    error: CRUDExample[0]
    error-message
    crit: CRUDExample[0]
    critical-message
    debug: Microsoft.Extensions.Hosting.Internal.Host[1]
    Hosting starting
    info: Microsoft.Hosting.Lifetime[1]
    Now Listening on: http://localhost:5298
    info: Microsoft.Hosting.Lifetime[1]
    Application started. Press Ctrl+C to shut down.
    info: Microsoft.Hosting.Lifetime[0]
    Hosting environment: Development
    info: Microsoft.Hosting.Lifetime[0]
    Content root path: C:\Users\Harsha\source\repos\CRUDSolution\CRUDExample\
    debug: Microsoft.Extensions.Hosting.Internal.Host[2]
    Hosting started
    debug: Microsoft.EntityFrameworkCore.Infrastructure[10401]
    An 'IServiceProvider' was created for internal use by Entity Framework.
    info: Microsoft.EntityFrameworkCore.Infrastructure[10403]

```

Predefined logs of Microsoft asp.net core



```

27
28 //create application pipeline
29 if (builder.Environment.IsDevelopment())
30 {
31     app.UseDeveloperExceptionPage();
32 }
33
34 app.Logger.LogDebug("debug-message");
35 app.Logger.LogInformation("information-message");
36 app.Logger.LogWarning("warning-message");
37 app.Logger.LogError("error-message");
38 app.Logger.LogCritical("critical-message");
39
40 if (builder.Environment.IsEnvironment("Test") == false)
41     Rotativa.AspNetCore.RotativaConfiguration.Setup(
42         wkhtmltopdfRelativePath: "Rotativa");
42

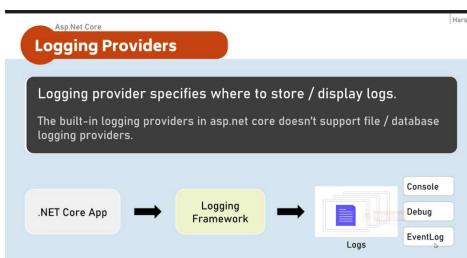
```

Okay, so these log messages are being written from the `Program.cs` file. But where do they actually go?

By default, ASP.NET Core writes logs to three places, known as **logging providers**:

- **Console** – Logs appear in the terminal or command prompt.
- **Debug** – Logs are written to the debug output (useful for debugging in Visual Studio).
- **Event Log** – On Windows, logs are stored in the Event Viewer.

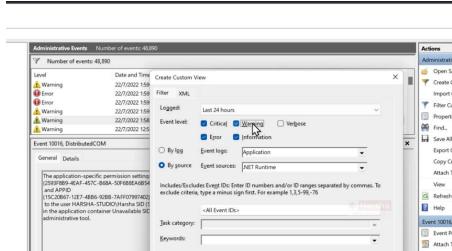
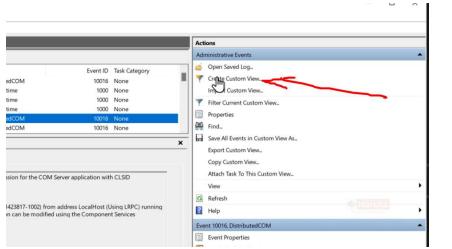
Additionally, any imported logging providers will also receive the log messages. This means when you log something in your .NET Core application, it automatically gets recorded in these locations unless you configure it otherwise.



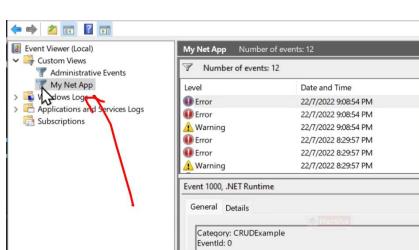
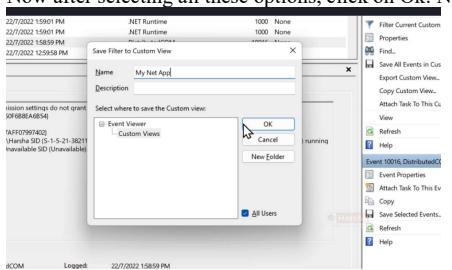
This is the application, so open up the same event viewer here. Okay, this is called event viewer, which can also be opened with control panel. Here, under custom views, you can find administrative views. Here, you can see all our log messages. Check for the log messages with Source equal to dot net runtime. For example, this one. See, you can see the errors as well as warnings. But the information or debug details are not logged here because the minimum log level for this event log is warning. So, all the warning, error, and critical logs are displayed here.

but you can find some other logs which are not generated by as these are automatically generated by the system for a different reason for different applications as well

In case if you want to filter out all remaining but only want to see your own application-specific logs, you can create a custom view. So, in the right-hand side, you can see 'Create custom view.' Click on that. In this 'Create custom view,' select only the last 24 hours. Because if you select any time, the list will be huge, so it takes much time to load. I am selecting the last 24 hours and by sorts. And whatever the source that you are interested in, only .NET runtime. So select it like this by checking this checkbox. And you would like to see all these: event level critical, error, information, warning.

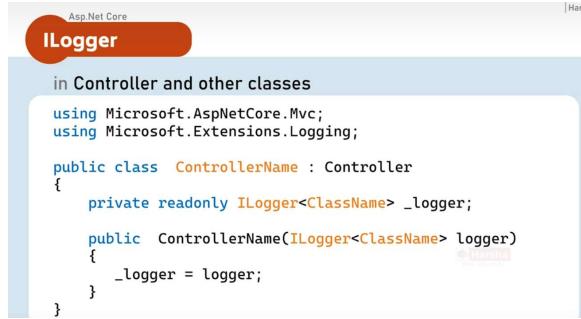


Now after selecting all these options, click on Ok. Name the view for example 'my dot net app'



Asp.Net Core ILogger in Controller, Service and Repository

How do you write actual logs? In every layer of the application, such as controllers, services, repositories, you have to leave some log messages which are useful for debugging as well as for tracking the execution sequence of the application. In order to do so, we have to inject a service of ILogger type, and we have to use that to log the messages.

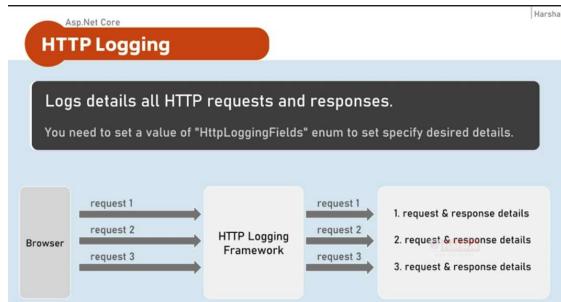


```
Asp.Net Core | Harsha
ILogger
in Controller and other classes
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;

public class ControllerName : Controller
{
    private readonly ILogger<ClassName> _logger;
    public ControllerName(ILogger<ClassName> logger)
    {
        _logger = logger;
    }
}
```

Asp.Net Core HTTP Logging

How do you track the incoming requests and corresponding HTTP responses? You can enable HTTP logging in your ASP.NET Core application so that it logs all the requests and corresponding response details in the log. HTTP logging is the process of recording the details of HTTP requests and responses as soon as they happen in real-time. For example, your application server has been started right now and it has received a particular request, let's call it as request 1. When the HTTP logging is enabled, it writes all the details of the HTTP request into the log. That means it writes the HTTP method, HTTP request path, HTTP request headers, etc. And also, as soon as we are sending the HTTP response from the server, it writes the response details as well in the log. So, we can track how many requests are being received, the code execution sequence, and also exception details and HTTP errors, if any. Fortunately, HTTP logging is a built-in feature of ASP.NET Core, you need not install any NuGet package for that.



```

Application started. Press Ctrl+C to shut down.
Microsoft.Hosting.Lifetime[0]
Hosting environment: Development
Microsoft.Hosting.Lifetime[0]
Content root path: C:\aspnetcore\CRUDSolution\CRUDExample\
Microsoft.Extensions.Hosting.Internal.Host[2]
Hosting started
Microsoft.AspNetCore.Hosting.Diagnostics[1]
Request starting HTTP/1.1 GET http://localhost:5298/ --
Microsoft.AspNetCore.HttpLogging.HttpLoggingMiddleware[1]
Request:
Protocol: HTTP/1.1
Method: GET
Scheme: http

```

Harsha



"By default, HTTP logging captures all details of the request and response, including request properties, response properties, request headers, and response headers. However, it excludes the request and response bodies, which are not logged by default. If needed, you can configure HTTP logging to capture only the specific details you want. The available options in the HTTP logging Fields enumeration allow you to customize what gets logged."

For example, the request method logs the HTTP method (GET, POST, PUT, etc.), while the request path logs the full URL path (e.g., <http://localhost:port/complete-path>). The request protocol logs the protocol used (e.g., HTTP/1 or HTTP/2), and the request scheme logs whether the request was made using HTTP or HTTPS. The request query logs any query strings in the request, and the request headers log all headers sent with the request. You can also choose to log the request body independently. By default, logging the request will include all these details: request properties, headers, and body.

Similarly, for the response, you can configure to log only response-specific details, such as status code and address. If you choose to log everything, both request and response details will be included. However, it's generally not recommended to log the request and response bodies due to performance concerns, as doing so can place a significant overhead on the server.

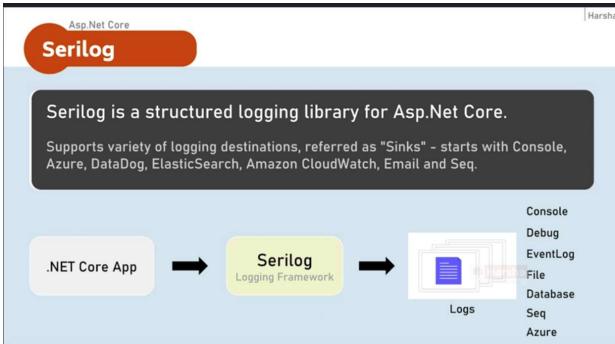
In general, the default logging configuration captures request and response properties and headers. You can adjust this by specifying the options you want to log, as we'll explore next."

Asp.Net Core	
"HttpLoggingFields" enum	
RequestMethod	Method of request. Eg: GET
RequestPath	Path of request. Eg: /home/index
RequestProtocol	Protocol of request. Eg: HTTP/1.1
RequestScheme	Protocol Scheme of request. Eg: http
RequestQuery	Query string Scheme of request. Eg: ?id=1
RequestHeaders	Headers of request. Eg: Connection: keep-alive
RequestPropertiesAndHeaders	Includes all of above (default)
RequestBody	Entire request body. [has performance drawbacks; not recommended]
Request	Includes all of above

Asp.Net Core	
"HttpLoggingFields" enum	
ResponseStatuscode	Status code of response. Eg: 200
ResponseHeaders	Headers of response. Eg: Content-Length: 20
ResponsePropertiesAndHeaders	Includes all of above (default)
ResponseBody	Entire response body. [has performance drawbacks; not recommended]
Response	Includes all of above
All	Includes all from request and response

Asp .Net Core

Serilog Basics

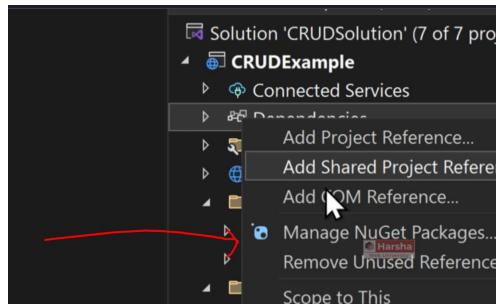


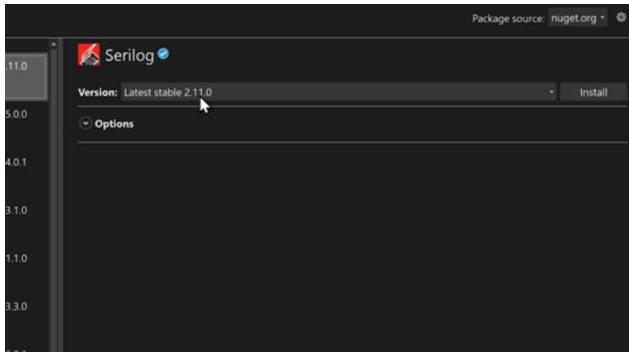
Serilog is one of the most popular logging frameworks in the ASP.NET Core ecosystem. But you might wonder, why use a third-party logging framework when we already have a built-in logging system in .NET Core?

The reason is that the built-in logging system has some limitations. For example, it doesn't support logging to external destinations like files or databases. This is where Serilog excels. As mentioned earlier, Serilog is the most commonly used logging framework in real-world .NET Core projects.

Serilog supports over 30 to 40 different logging destinations, known as 'sinks.' These sinks specify where log messages should be stored. For instance, the 'console' sink outputs log messages to the console window. Similar to the built-in logging system, Serilog also supports debug and event log sinks. However, Serilog goes a step further by offering a wide variety of other logging destinations, such as file storage, databases, and cloud services.

Some of the supported databases include PostgreSQL, CouchDB, SQL Server, MySQL, and more. Serilog also integrates with services like Datadog, Elasticsearch, Amazon Web Services (AWS), and Azure. In this lesson, we'll get started with Serilog and explore how to install and configure it in your .NET Core application.

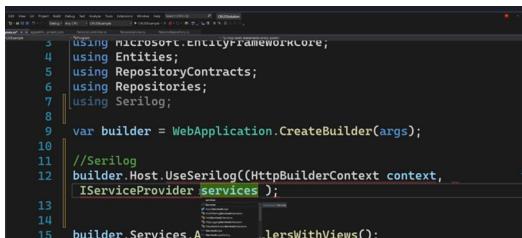




It receives the context of the asp.net core application builder, using which you can access the asp.net core Application configuration or services.

```
8 var builder = WebApplication.CreateBuilder(args);
9
10 //Serilog
11 builder.Host.UseSerilog((HttpBuilderContext context, );
12
13
14 builder.Services.AddControllersWithViews();
15
16 //Add services to the container.
```

The second one is **IServiceProvider**, this parameter represents the service collection that is built in IOC container in asp.net core application.



```
3 using Microsoft.EntityFrameworkCore;
4 using Entities;
5 using RepositoryContracts;
6 using Repositories;
7 using Serilog;
8
9 var builder = WebApplication.CreateBuilder(args);
10
11 //Serilog
12 builder.Host.UseSerilog((HttpBuilderContext context,
13 IServiceProvider services );
14
15 builder.Services.AddControllersWithViews();
```

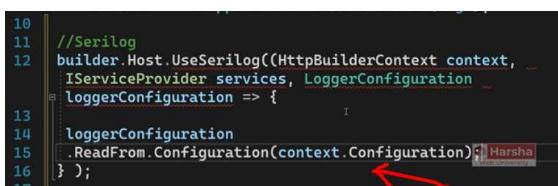
This will represent the configuration of the serilog.



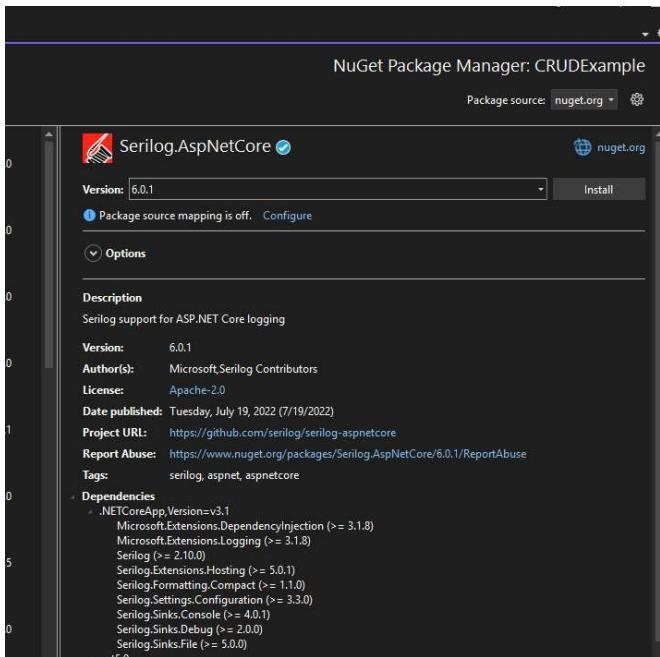
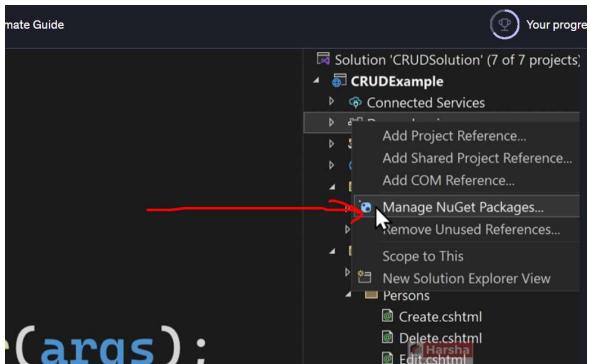
```
11 //Serilog
12 builder.Host.UseSerilog((HttpBuilderContext context,
13 IServiceProvider services, ILoggerConfiguration
14 loggerConfiguration );
```

Hey logger configuration, I would like to read from configuration, It means that, we are reading from configuration settings from our asp.net core application from the host builder and assigning the same configuration into the serilog.

Now the serilog package is able to read the configuration settings of our appSettings.json or appSettings.Development.json based on the environment.



```
10 //Serilog
11 builder.Host.UseSerilog((HttpBuilderContext context,
12 IServiceProvider services, ILoggerConfiguration
13 loggerConfiguration => {
14
15 loggerConfiguration
16 .ReadFrom.Configuration(context.Configuration);
```



Now open up appSettings.json file

```

1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft.AspNetCore": "Warning"
6      }
7    },
8    "AllowedHosts": "*",
9    "ConnectionStrings": {
10      "DefaultConnection": "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=PersonsDatabase;Integrated Security=True;Connect"
11    }
12  }
13

```

We don't need it anymore.

Serilog - Configuration

in appsettings.json

```
{
  "Serilog": {
    "Using": [
      "Serilog.Sinks.YourSinkHere"
    ],
    "MinimumLevel": "Debug | Information | Warning | Error | Critical",
    "WriteTo": [
      {
        "Name": "YourSinkHere",
        "Args": "YourArguments"
      }
    ]
  }
}
```



appsettings.json X Program.cs

Schema: <https://json.schemastore.org/appsettings.json>

```

1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft.AspNetCore": "Warning"
6      }
7    },
8    "AllowedHosts": "*",
9    "ConnectionStrings": {
10      "DefaultConnection": "Data Source=(local)\SQLEXPRESS;Initial Catalog=ExcelDB;Integrated Security=True"
11    },
12    "EPPlus": {
13      "ExcelPackage": {
14        "LicenseContext": "NonCommercial"
15      }
16    },
17    "Serilog": {
18      "MinimumLevel": "Debug",
19      "Using": [
20        "Serilog.Sinks.Console"
21      ],
22      "WriteTo": [
23        {
24          "Name": "Console"
25        }
26      ]
27    }
28  }
29

```

```

[22:45:28 DBG] Initializing Razor view compiler with compiled view: '/Views/Countries/UploadFromExcel.cshtml'.
[22:45:29 DBG] Initializing Razor view compiler with compiled view: '/Views/Persons/Create.cshtml'.
[22:45:29 DBG] Initializing Razor view compiler with compiled view: '/Views/Persons/Delete.cshtml'.
[22:45:29 DBG] Initializing Razor view compiler with compiled view: '/Views/Persons/Edit.cshtml'.
[22:45:29 DBG] Initializing Razor view compiler with compiled view: '/Views/Persons/Index.cshtml'.
[22:45:29 DBG] Initializing Razor view compiler with compiled view: '/Views/Persons/PersonsPDF.cshtml'.
[22:45:29 DBG] Initializing Razor view compiler with compiled view: '/Views/Shared/_GridColumnHeader.cshtml'.
[22:45:29 DBG] Initializing Razor view compiler with compiled view: '/Views/Shared/_Layout.cshtml'.
[22:45:29 DBG] Initializing Razor view compiler with compiled view: '/Views/_ViewImports.cshtml'.
[22:45:29 DBG] Initializing Razor view compiler with compiled view: '/Views/_ViewStart.cshtml'.
[22:45:29 DBG] Registered model binder providers, in the following order: ["Microsoft.AspNetCore.Mvc.ModelBinding.BindersBinderTypeModelBinderProvider", "Microsoft.AspNetCore.Mvc.ModelBinding.Binders.ServicesModelBinderProvider", "Microsoft.AspNetCore.Mvc.ModelBinding.Binders.BodyModelBinderProvider", "Microsoft.AspNetCore.Mvc.ModelBinding.Binders.FloatingPointTypeModelBinderProvider", "Microsoft.AspNetCore.Mvc.ModelBinding.Binders.EnumTypeModelBinderProvider", "Microsoft.AspNetCore.Mvc.ModelBinding.Binders.SimpleTypeModelBinderProvider", "Microsoft.AspNetCore.Mvc.ModelBinding.Binders.CancellationTokenModelBinderProvider", "Microsoft.AspNetCore.Mvc.ModelBinding.Binders.ByteArrayModelBinderProvider", "Microsoft.AspNetCore.Mvc.ModelBinding.Binders.FormFileModelBinderProvider", "Microsoft.AspNetCore.Mvc.ModelBinding.Binders.FormCollectionModelBinderProvider", "Microsoft.AspNetCore.Mvc.ModelBinding.Binders.KeyValuePairModelBinderProvider", "Microsoft.AspNetCore.Mvc.ModelBinding.Binders.DictionaryModelBinderProvider", "Microsoft.AspNetCore.Mvc.ModelBinding.Binders.CollectionModelBinderProvider", "Microsoft.AspNetCore.Mvc.ModelBinding.Binders.ComplexObjectModelBinderProvider"]
[22:45:29 DBG] Hosting starting
[22:45:29 INF] User profile is available. Using 'C:\Users\Harsha\AppData\Local\ASP.NET\DataProtection-Keys' as key repository and Windows DPAPI to encrypt keys at rest.
[22:45:29 DBG] Reading data from file 'C:\Use NOW you can see that .NET\DataProtection-Keys\key-cb8bf7b2-3a5f-4d11-bb1c-a1295bb17212.xml'!

```

Asp.Net Core Serilog File Sink

Asp.Net Core

Harsha

Serilog - File Sink

The "Serilog.Sinks.File" logs into a specified file.

You can configure the filename, rolling interval, file size limit etc., using configuration settings.

.NET Core App



Serilog
Logging Framework



File Logs

Asp.Net Core Serilog Database Sink

Serilog - MSSqlServer Sink

The "Serilog.Sinks.MSSqlServer" logs into a specified SQL Server database.

You can configure the connection string using configuration settings.

.NET Core App

Serilog
Logging Framework

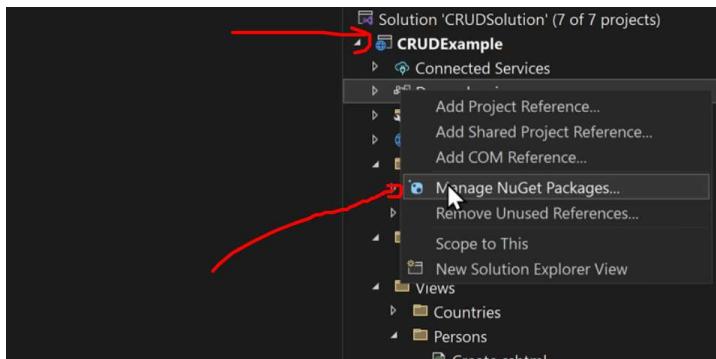


SQL Server
Database Logs

"You can configure Serilog to log directly to an SQL Server database. Serilog not only supports file-based logging but also integrates with various databases such as MS SQL Server, PostgreSQL, MySQL, and Cosmos DB, among others.

In this lesson, we'll focus on configuring MS SQL Server as a sink for Serilog. Our goal is to store and view Serilog logs within an SQL Server database.

To get started, the first step is to install the MS SQL Server sink for Serilog.



Solution 'CRUDSolution' (7 of 7 projects)

CRUDExample

Connected Services

Add Project Reference...

Add Shared Project Reference...

Add COM Reference...

Manage NuGet Packages...

Remove Unused References...

Scope to This

New Solution Explorer View

Views

Countries

Persons

Create solution

Serilog.Sinks.MSSqlServer

Version: 5.7.1

Install

Description

A Serilog sink that writes events to Microsoft SQL Server

Version: 5.7.1

Author(s): Michiel van Oudheusden, Christian Kadluba, Serilog Contributors

License: Apache-2.0

Downloads: 9,987,378

Date published: Saturday, June 18, 2022 (6/18/2022)

Project URL: <https://github.com/serilog-mssql/serilog-sinks-mssqlserver>

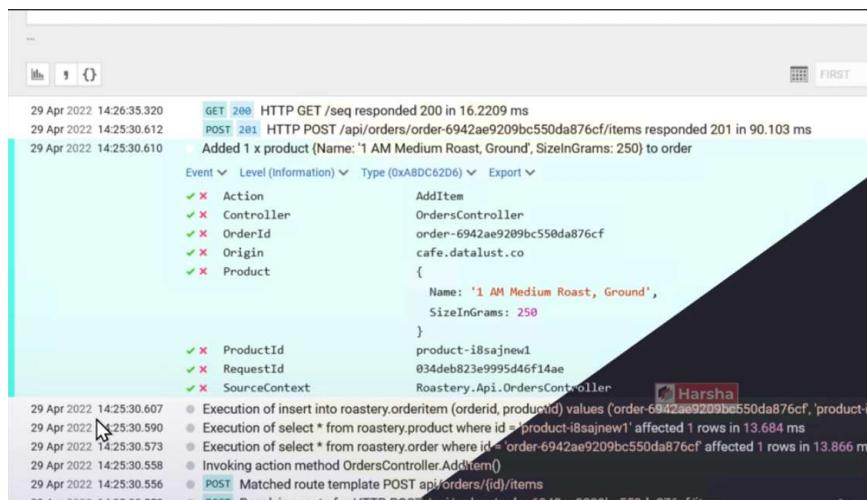
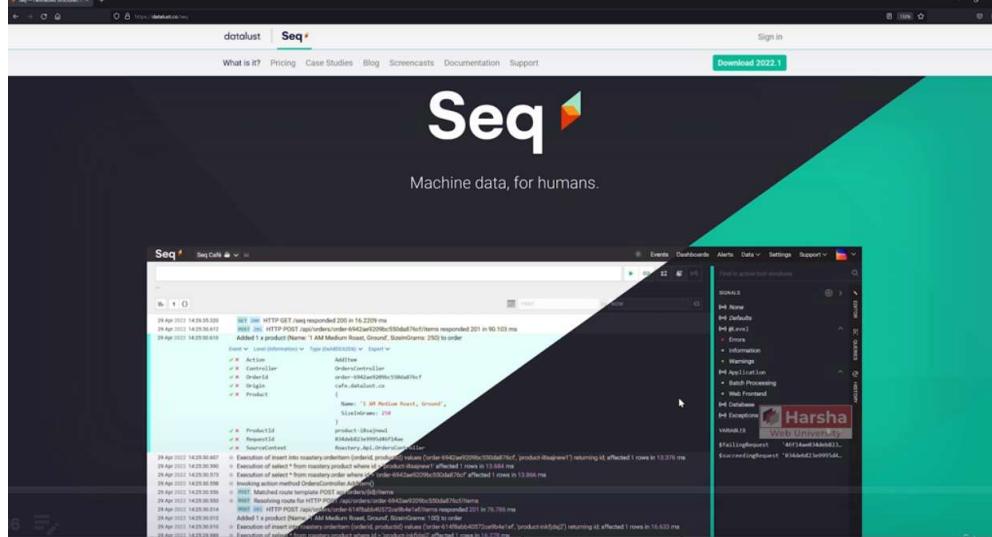
Report Abuse: <https://www.nuget.org/packages/Serilog.Sinks.MSSqlServer/5.7.1/ReportAbuse>

Tags: serilog, sinks, mssqlserver

Dependencies

Asp.Net Core

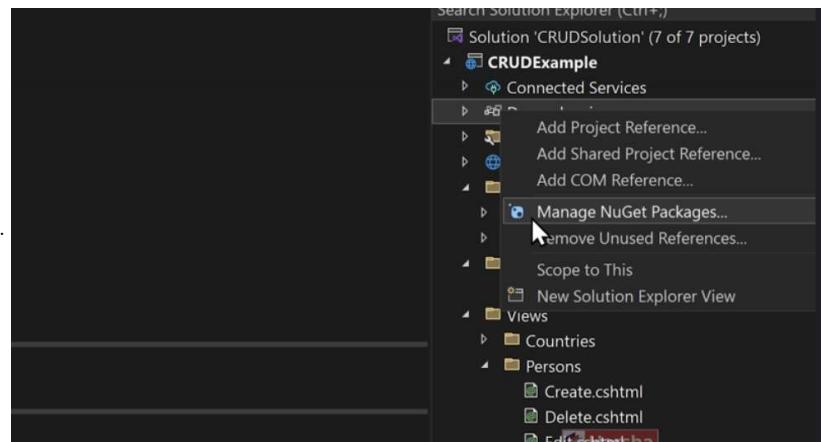
Serilog Seq



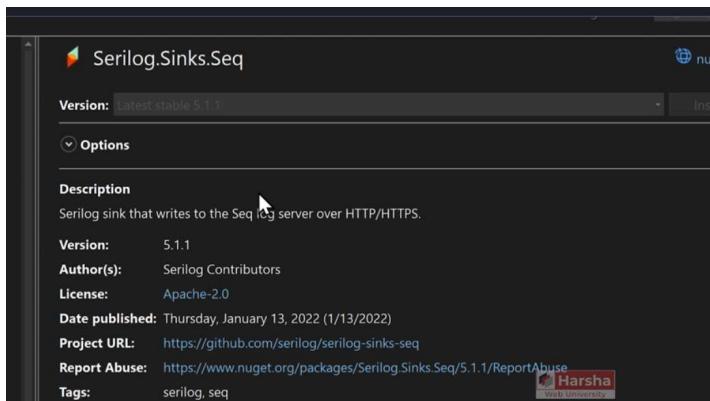
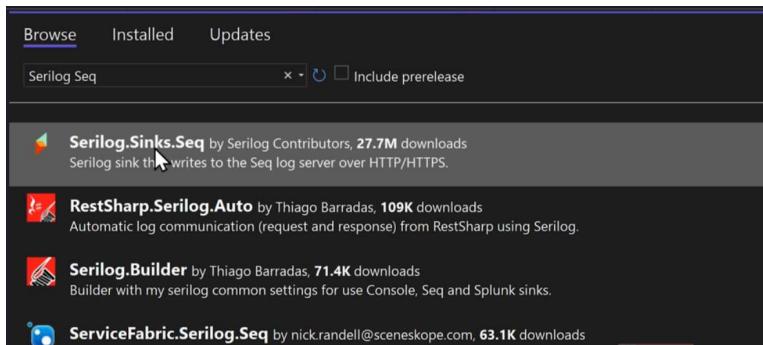
The primary reason for using Seq is that it provides a powerful monitoring tool to track application logs in real time. This allows developers to observe logs being generated for each live request directly from the server.

For example, Seq offers an intuitive interface where you can view logs with timestamps, log levels, and structured messages. By default, Seq supports structured logging, making it easier to filter, search, and analyze logs efficiently.

1. Download Seq



3.



appsettings.json

```

31     "Args": {
32         "path": "logs/log.txt",
33         "rollingInterval": "Hour", //every one hour a new file gets created
34         "fileSizeLimitBytes": 1048576, //whenever the file size exceeds 1MB, even though
35         "rollOnFileSizeLimit": true //as soon as the 1mb limit is reached for the log
36     },
37 },
38     "Name": "MSSqlServer",
39     "Args": {
40         "connectionString": "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=CRUDExample",
41         "tableName": "Logs",
42         "autoCreateSqlTable": true
43     }
44 },
45     {
46         "Name": "Seq",
47         "Args": {
48             "serverUrl": "http://localhost:5341"
49         }
50     }
51 ],
52 }
53 ]
54 }
55 
```

Fluid Assertion Methods | abhikhalid/Asp.Net-Core-9.N | Course: Asp.Net Core 9 (.NET 9) | Events — Seq

localhost:5341/#/events?range=1d&tail

Events | Dashboards | Alerts | Data | Settings | Support | Find in active tool windows

SIGNALS | QUERIES | VARIABLES | HISTORY

Last 1d

24 Feb 2025 18:19:50.305 Request finished HTTP/1.1 GET http://localhost:5195/_vs/browserLink - 200 null text/javascript; charset=UTF-8
24 Feb 2025 18:19:50.230 Request starting HTTP/1.1 GET http://localhost:5195/_vs/browserLink - null null
24 Feb 2025 18:19:50.197 Request finished HTTP/1.1 GET http://localhost:5195/_framework/aspnetcore-browser-refresh.js - 200 137...
24 Feb 2025 18:19:50.188 Request starting HTTP/1.1 GET http://localhost:5195/_framework/aspnetcore-browser-refresh.js - null null
24 Feb 2025 18:19:50.144 Request finished HTTP/1.1 GET http://localhost:5195/_ - 200 null text/html; charset=UTF-8 1324.2249ms
24 Feb 2025 18:19:50.132 Response: StatusCode: 200 Content-Type: text/html; charset=UTF-8 Date: Mon, 24 Feb 2025 12:19:49 GMT S...
24 Feb 2025 18:19:50.128 Executed endpoint 'CRUDExample.Controllers.PersonsController.Index (CRUDExample)'
24 Feb 2025 18:19:50.126 Executed action CRUDExample.Controllers.PersonsController.Index (CRUDExample) in 1196.8805ms
24 Feb 2025 18:19:50.122 Executed ViewResult - view Index executed in 189.9618ms.
24 Feb 2025 18:19:49.933 Executing ViewResult, running view Index.
24 Feb 2025 18:19:49.911 GetSortedPersons of PersonService
24 Feb 2025 18:19:49.780 Executed DbCommand (25ms) [Parameters=@[], CommandType='Text', CommandTimeout='30'] SELECT [p].[P...
24 Feb 2025 18:19:49.112 GetAllPersons of PersonsRepository
24 Feb 2025 18:19:49.111 GetFilteredPersons of PersonService
24 Feb 2025 18:19:49.081 Index action method of PersonsController
24 Feb 2025 18:19:48.925 Route matched with (action = "Index", controller = "Persons"). Executing controller action with signature Syst...
24 Feb 2025 18:19:48.881 Executing endpoint 'CRUDExample.Controllers.PersonsController.Index (CRUDExample)'
24 Feb 2025 18:19:48.879 Request: Protocol: HTTP/1.1 Method: GET Scheme: http PathBase: Path: /
24 Feb 2025 18:19:48.829 Request starting HTTP/1.1 GET http://localhost:5195/ - null null
24 Feb 2025 18:19:45.835 Content root path: D:\ASSIGNMENT\Practice\DOT NET CORE\Asp.Net-Core-9-.NET-9-True-Ultimate-Guide\...
24 Feb 2025 18:19:45.834 Hosting environment: Development
24 Feb 2025 18:19:45.824 Application started. Press Ctrl+C to shut down.
24 Feb 2025 18:19:45.784 Now listening on: http://localhost:5195

2024.3.13545 Individual License

Asp.Net Core

Serilog RequestId

request ID is the unique number that is generated automatically for each request so that you can write all the logs of the request with reference to the a request ID
let me show that

Asp.Net Core

| Harsha

Serilog - RequestId

"RequestId" is the unique number (guid) of each individual requests, used to identify to which request the log belongs to.

RequestId is "Tracelocator" internally, that is generated by Asp.Net Core.



Event	Level (Information)	Type (0x702B74E1)	Export
✓ ✘ ConnectionId	0HMJEB60BC5UC		
✓ ✘ ContentLength			
✓ ✘ ContentType			
✓ ✘ EventId	{Id: 1}		
✓ ✘ Host	localhost:5298		
✓ ✘ HostingRequestStartingLog	Request starting HTTP/1.1 GET http://localhost:5298/		
✓ ✘ Method	GET		
✓ ✘ Path	/		
✓ ✘ PathBase			
✓ ✘ Protocol	HTTP/1.1		
✓ ✘ QueryString			
✓ ✘ RequestId	0HMJEB60BC5UC:00000002		
✓ ✘ RequestPath	/		
✓ ✘ Scheme	http	so this is the request ID that is	

GetAllPersons of PersonsRepository			
GetFilteredPersons of PersonsService			
Index action method of PersonsController			
Route matched with {action= "Index", controller = "Persons"}. Executing endpoint 'CRUDEExample.Controllers.PersonsController.Index'			
Request: Protocol: HTTP/1.1 Method: GET Scheme: http PathBase: Pa			
Request starting HTTP/1.1 GET http://localhost:5298/ --			
Event	Level (Information)	Type (0x1C2B74E1)	Export

✗ ActionId	f7627bf4-725e-4af2-8b3f-08f7691e
✗ ActionName	CRUDEExample.Controllers.PersonsCo
✗ AssemblyName	CRUDEExample
✗ ConnectionId	0HMJEB60BC5UC
✗ Controller	CRUDEExample.Controllers.PersonsCo
✗ EventId	{Id: 3, Name: 'ControllerActionB
✗ MethodInfo	System.Threading.Tasks.Task`1[Mi
	lt] Index(System.String, System.
	ServiceContracts.Enums.SortOrder
✗ RequestId	0HMJEB60BC5UC:00000002 ↪
✗ RequestPath	/

Asp.Net Core Serilog Enrichers

Asp.Net Core

Harsha

Serilog - Enrichers

Enrichers are additional details that are added to LogContext; so they're stored in logs.

Eg: MachineName [or] Custom Properties.



In **Serilog**, enrichers are additional values that get logged alongside your log messages.

For instance, when you log a message from your **ASP.NET Core** application, **Serilog** automatically appends extra details to the log context. These additional details are then stored in the final logging destination (sink).

Enrichers can include predefined values like:

- **Machine Name** (Server Name)
- **Thread ID**
- **Request ID**
- **User Information**

Additionally, you can define **custom properties** in your `appsettings.json`, which Serilog will include in every log entry.

Now, let me show you how to configure this in **Serilog**.

Asp.Net Core

Serilog IDiagnosticContext

Diagnostic Context in Serilog

The **diagnostic context** is an advanced way to add extra property values to log entries in **Serilog**.

Unlike the **log context**, which adds properties to every log within a request, the **diagnostic context** only logs properties once—at the end of the request.

Key Differences Between Log Context and Diagnostic Context:

1. Log Context:

- Adds properties to **every log entry** within the request.
- Useful for tracking values across multiple logs.

2. Diagnostic Context:

- Logs properties **only once**, in the **final log entry** of the request.

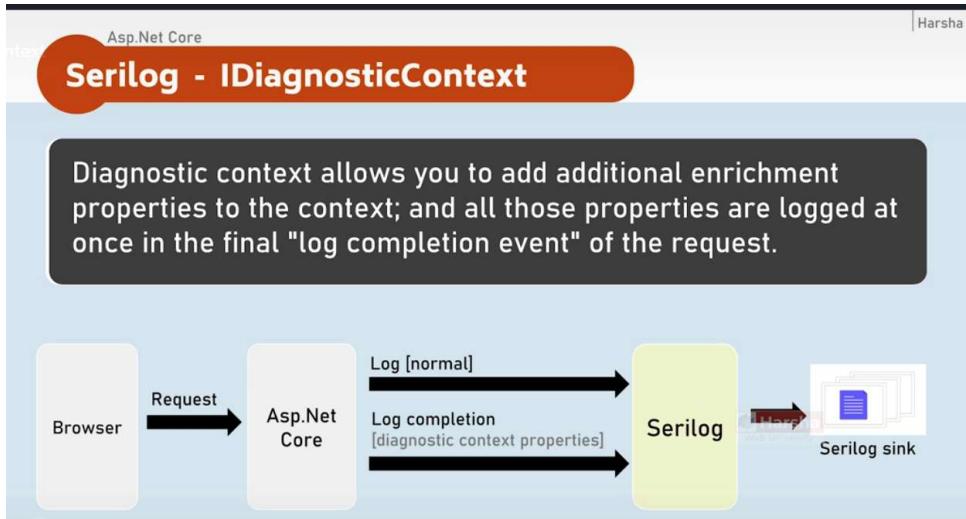
- Ideal for capturing **summary details** about the request.
- Properties can be added dynamically at any layer (Controller, Service, Repository, etc.).
- Each request gets its **own diagnostic context**, identified by a **Request ID**.

Example Use Case:

Imagine an API request that generates **10 logs** during processing.

- If you use **Log Context**, all 10 logs will include the same extra properties.
- If you use **Diagnostic Context**, only the **last log (completion log)** will include those properties.

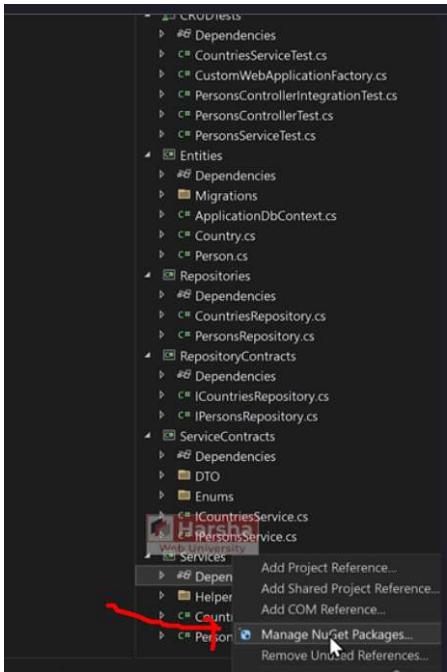
This approach helps avoid redundant logging while still capturing crucial information at the **end of the request lifecycle**.



I would like to see the list of persons in the log.

```

5
6
7 7 references
8
9 public async Task<List<PersonResponse>> GetAllPersons()
10 {
11     _logger.LogInformation("GetAllPersons of PersonsService");
12
13     var persons = await _personsRepository.GetAllPersons();
14
15     return persons
16     .Select(temp => temp.ToPersonResponse()).ToList();
17 }
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
305
306
307
308
309
309
310
311
312
313
313
314
315
316
316
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
```



Serilog.Extensions.Hosting

Version: 5.0.1

Install

Options

Description

Serilog support for .NET Core logging in hosted services

Version: 5.0.1

Author(s): Microsoft,Serilog Contributors

License: Apache-2.0

Downloads: 115,138,588

Date published: Tuesday, July 19, 2022 (7/19/2022)

Project URL: <https://github.com/serilog/serilog-extensions-hosting>

Report Abuse: <https://www.nuget.org/packages/Serilog.Extensions.Hosting/5.0.1/ReportAbuse>

Tags: serilog, aspnet, aspnetcore, hosting

Dependencies

- .NETStandard.Version=v2.0
 - Microsoft.Extensions.DependencyInjection.Abstractions (>= 3.1.8)
 - Microsoft.Extensions.Hosting.Abstractions (>= 3.1.8)
 - Microsoft.Extensions.Logging.Abstractions (>= 3.1.8)

24 Feb 2025 21:09:23.557	Request finished HTTP/1.1 GET http://localhost:5195/_framework/aspnetcore-browser-refresh.js - 200 13774 application/javascript; charset=utf-8 0.411ms																														
24 Feb 2025 21:09:23.550	Request starting HTTP/1.1 GET http://localhost:5195/_framework/aspnetcore-browser-refresh.js - null null																														
24 Feb 2025 20:24:15:09:23.550Z	Request finished HTTP/1.1 GET http://localhost:5195/-200 null text/html; charset=utf-8 1173.6218ms																														
24 Feb 2025 21:09:23.485	HTTP GET / responded 200 in 1058.2142 ms																														
<p>Event ▾ Level (Information) ▾ Type (0x37AA1435) ▾ Export ▾</p> <table border="1"> <thead> <tr> <th>Event</th> <th>Level</th> <th>Type</th> </tr> </thead> <tbody> <tr> <td>✓ ✘ ApplicationName</td> <td>CRUD Demp App</td> <td>0HNAKSKR8HNTM</td> </tr> <tr> <td>✓ ✘ ConnectionId</td> <td></td> <td></td> </tr> <tr> <td>✓ ✘ Elapsed</td> <td>1058.2142</td> <td></td> </tr> <tr> <td>✓ ✘ Persons</td> <td>['Entities.Person', 'Entities.Person']</td> <td></td> </tr> <tr> <td>✓ ✘ RequestId</td> <td>0HNAKSKR8HNTM:00000001</td> <td></td> </tr> <tr> <td>✓ ✘ RequestMethod</td> <td>GET</td> <td></td> </tr> <tr> <td>✓ ✘ RequestPath</td> <td>/</td> <td></td> </tr> <tr> <td>✓ ✘ SourceContext</td> <td>Serilog.AspNetCore.RequestLoggingMiddleware</td> <td></td> </tr> <tr> <td>✓ ✘ StatusCode</td> <td>200</td> <td></td> </tr> </tbody> </table>		Event	Level	Type	✓ ✘ ApplicationName	CRUD Demp App	0HNAKSKR8HNTM	✓ ✘ ConnectionId			✓ ✘ Elapsed	1058.2142		✓ ✘ Persons	['Entities.Person', 'Entities.Person']		✓ ✘ RequestId	0HNAKSKR8HNTM:00000001		✓ ✘ RequestMethod	GET		✓ ✘ RequestPath	/		✓ ✘ SourceContext	Serilog.AspNetCore.RequestLoggingMiddleware		✓ ✘ StatusCode	200	
Event	Level	Type																													
✓ ✘ ApplicationName	CRUD Demp App	0HNAKSKR8HNTM																													
✓ ✘ ConnectionId																															
✓ ✘ Elapsed	1058.2142																														
✓ ✘ Persons	['Entities.Person', 'Entities.Person']																														
✓ ✘ RequestId	0HNAKSKR8HNTM:00000001																														
✓ ✘ RequestMethod	GET																														
✓ ✘ RequestPath	/																														
✓ ✘ SourceContext	Serilog.AspNetCore.RequestLoggingMiddleware																														
✓ ✘ StatusCode	200																														

Asp.Net Core

Serilog Timings

The **Serilog Timings** is another package that works on top of **Serilog**. It is one of the best ways to record the time taken for the execution of a particular block of code and add the same into the logs. For example, there is a method that contains a massive amount of code, or a method that connects to the database, or a method that contains a large number of loops.

Suppose you feel that, in general, there is a possibility that it takes a significant amount of time—maybe **a few milliseconds** or **a few seconds**. But you would like to measure **exactly** how much time it takes for actual execution—whether it is a few milliseconds or seconds and how many exactly.

You would like to know these details in the logs.

Exactly for this requirement, you will use the **Serilog Timings** package, wherever you feel your code may take much time to execute.

Asp.Net Core | Harsha

Serilog Timings

"SerilogTimings" package records timing of a piece of your source code, indicating how much time taken for executing it.

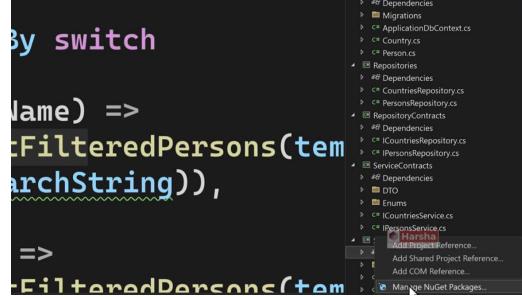
SerilogTimings - Start
Time Taken: 00:01 sec.
SerilogTimings - End

Let's say in our PersonService.cs file, we have 'GetFilteredPerson' method.

```
public async Task<List<PersonResponse>> GetFilteredPersons(string searchBy, string? searchString)
{
    _logger.LogInformation("GetFilteredPersons of PersonService");

    List<Person> persons = searchBy switch {
        nameof(PersonResponse.PersonName) =>
            await _personsRepository.GetFilteredPersons(temp => temp.PersonName.Contains(searchString)),
        nameof(PersonResponse.Email) =>
            await _personsRepository.GetFilteredPersons(temp => temp.Email.Contains(searchString)),
        nameof(PersonResponse.DateOfBirth) =>
            await _personsRepository.GetFilteredPersons(temp => temp.DateOfBirth.Value.ToString("dd MMMM yyyy").Contains(searchString)),
        nameof(PersonResponse.Gender) =>
            await _personsRepository.GetFilteredPersons(temp => temp.Gender.Contains(searchString)),
        nameof(PersonResponse.CountryID) =>
            await _personsRepository.GetFilteredPersons(temp =>
```

Click on services and select 'manage nugget packages'



NuGet Package Manager: Services

Package source: nuget.org

SerilogTimings

Version: 3.0.1

Install

Package source mapping is off. Configure

Options

Description

Extend Serilog with timed operations.

Version: 3.0.1

Author(s): nblumhardt,SerilogTimings Contributors

License: Apache-2.0

Date published: Saturday, July 16, 2022 (7/16/2022)

Project URL: <https://github.com/nblumhardt/serilog-timings>

Report Abuse: <https://www.nuget.org/packages/SerilogTimings/3.0.1/ReportAbuse>

Tags: serilog, metrics, timings, operations

Dependencies

- net6.0
 - Serilog (>= 2.10.0)
 - .NETStandard,Version=v2.0
 - Serilog (>= 2.10.0)

24 Feb 2025 21:37:40.085	Executed ViewResult - view Index executed in 287.7972ms.																																								
24 Feb 2025 21:37:39.798	Executing ViewResult, running view Index.																																								
24 Feb 2025 21:37:39.775	GetSortedPersons of PersonService																																								
24 Feb 2025 21:37:39.751	Time for Filtered Persons from Database completed in 1536.0 ms																																								
	<table border="1"> <thead> <tr> <th>Event</th> <th>Level (Information)</th> <th>Type (0x29BAE5A9)</th> <th>Export</th> </tr> </thead> <tbody> <tr> <td>✓ ✘ ActionId</td> <td></td> <td>fbc3c608-6138-4e62-b29b-306b72ff5410</td> <td></td> </tr> <tr> <td>✓ ✘ ActionName</td> <td></td> <td>CRUDExample.Controllers.PersonsController.Index (CRUDExample)</td> <td></td> </tr> <tr> <td>✓ ✘ ApplicationName</td> <td></td> <td>CRUD Demp App</td> <td></td> </tr> <tr> <td>✓ ✘ ConnectionId</td> <td></td> <td>0HNAKT4L2Q3BK</td> <td></td> </tr> <tr> <td>✓ ✘ Elapsed</td> <td></td> <td>1535.9618</td> <td></td> </tr> <tr> <td>✓ ✘ OperationId</td> <td></td> <td>06413c38-fa00-4890-880f-f902c3bdd8f5</td> <td></td> </tr> <tr> <td>✓ ✘ Outcome</td> <td></td> <td>completed</td> <td></td> </tr> <tr> <td>✓ ✘ RequestId</td> <td></td> <td>0HNAKT4L2Q3BK:00000001</td> <td></td> </tr> <tr> <td>✓ ✘ RequestPath</td> <td></td> <td>/</td> <td></td> </tr> </tbody> </table>	Event	Level (Information)	Type (0x29BAE5A9)	Export	✓ ✘ ActionId		fbc3c608-6138-4e62-b29b-306b72ff5410		✓ ✘ ActionName		CRUDExample.Controllers.PersonsController.Index (CRUDExample)		✓ ✘ ApplicationName		CRUD Demp App		✓ ✘ ConnectionId		0HNAKT4L2Q3BK		✓ ✘ Elapsed		1535.9618		✓ ✘ OperationId		06413c38-fa00-4890-880f-f902c3bdd8f5		✓ ✘ Outcome		completed		✓ ✘ RequestId		0HNAKT4L2Q3BK:00000001		✓ ✘ RequestPath		/	
Event	Level (Information)	Type (0x29BAE5A9)	Export																																						
✓ ✘ ActionId		fbc3c608-6138-4e62-b29b-306b72ff5410																																							
✓ ✘ ActionName		CRUDExample.Controllers.PersonsController.Index (CRUDExample)																																							
✓ ✘ ApplicationName		CRUD Demp App																																							
✓ ✘ ConnectionId		0HNAKT4L2Q3BK																																							
✓ ✘ Elapsed		1535.9618																																							
✓ ✘ OperationId		06413c38-fa00-4890-880f-f902c3bdd8f5																																							
✓ ✘ Outcome		completed																																							
✓ ✘ RequestId		0HNAKT4L2Q3BK:00000001																																							
✓ ✘ RequestPath		/																																							
24 Feb 2025 21:37:39.587	Executed DbCommand (32ms) [Parameters=@P1, CommandType='Text', CommandTimeout='30'] SELECT @P1 AS [PersonID], @P1 AS [Address], @P1 AS [CountryID]																																								