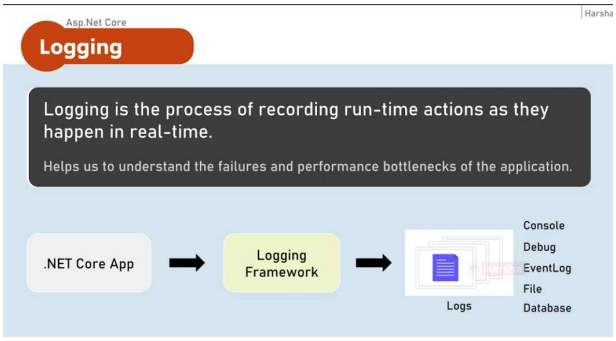
Section 20: Logging and Serilog

Sunday, February 23, 2025 8:21 PM



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.

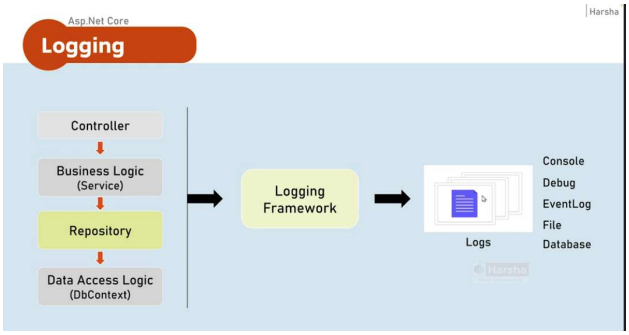
Logging Frameworks & Logs

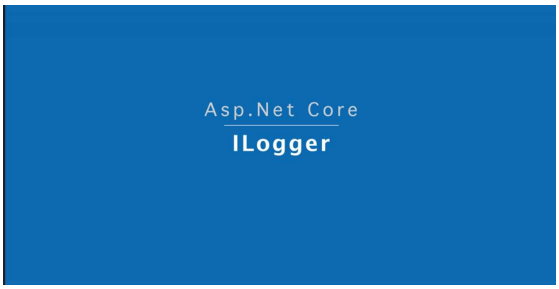
- The **logging framework** acts as a **mediator** between your **application layers** and the **log storage**.
- **Logs** are data stores where log messages are **persisted** for later review.
- **Console logs** are temporary and immediately displayed in the **Kestrel window**.
- Other logs can be stored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

Using Serilog for Logging

In this section, we'll learn how to:

- ☑ **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**.





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 the 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.");
}
else
{
    _logger.LogWarning("Unexpected behavior: isDeleted is false.");
}
```

📌 **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
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("wwwroot",
      wkhtmltopdfRelativePath: "Rotativa");
42
43  app.UseStaticFiles();
```

Run and see the output in the kestrel window.

```
info: CRUDExample[0]
      information-message
warn: CRUDExample[0]
      warning-message
fail: CRUDExample[0]
      error-message
crit: CRUDExample[0]
      critical-message
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5195
info: 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\DOT NET CORE\~Asp.Net-Core-9~.NET-9-True-Ultimate-Guide\Section_20_Logging and Serilog\CRUDExample
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (60ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT [p].[PersonID], [p].[Address], [p].[CountryID], [p].[DateOfBirth], [p].[Email], [p].[Gender], [p].[PersonName], [p].[ReciveNewsLetters], [p].[TaxIdentificationNumber], [c].[CountryID], [c].[Countr
yName]
      FROM [Persons] 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

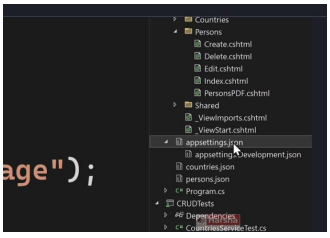
ASP.NET Core supports the following log levels, ranked by severity:

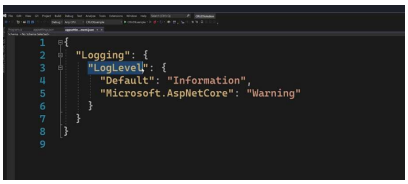
- Trace (rarely used in real projects)
- Debug (for tracking variable values and debugging)
- Information (for general execution flow, e.g., tracking method calls)
- Warning (for unexpected but non-breaking issues, like unexpected conditions)
- Error (for exceptions and runtime errors)
- Critical (for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

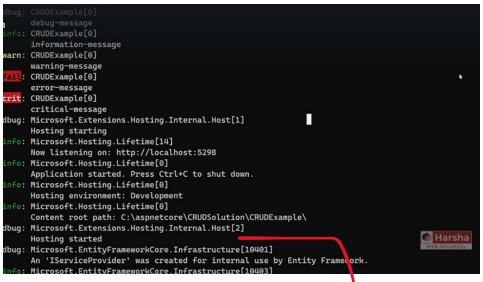
By default, logs below Information (such as Debug and Trace) are ignored.

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.





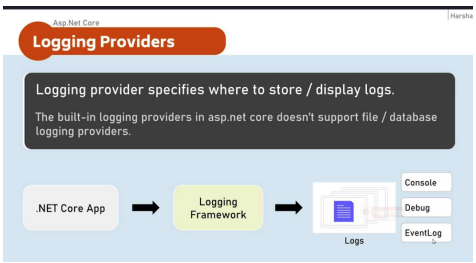


Predefined logs of Microsoft asp.net core

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**:
1. **Console** – Logs appear in the terminal or command prompt.
2. **Debug** – Logs are written to the debug output (useful for debugging in Visual Studio).
3. **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.



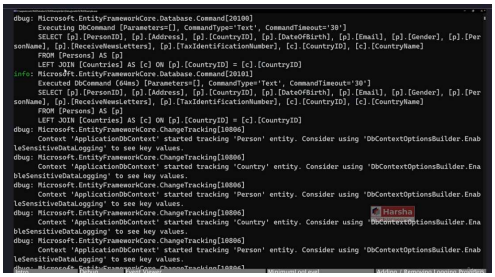
There is also another logging model called **Event Source**, but it's not commonly used in real-world projects. So, we'll focus on the three main logging providers: **Console, Debug, and Event Log**.
Here, we are writing log messages for five different log levels:

- **Debug**
- **Information**
- **Warning**
- **Error**
- **Critical**

You can log any number of messages using any log level.
Now, run the application.

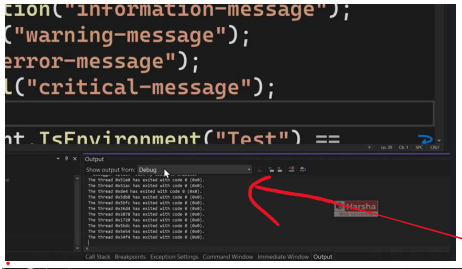
Viewing Console Logs

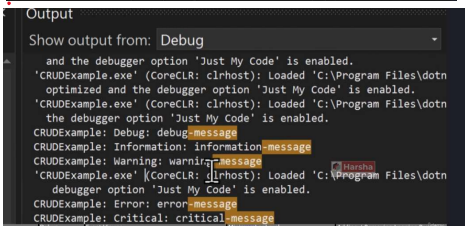
Since the first log provider is **Console**, you can see these logs in the **command prompt with Kestrel**. This window opens automatically when you start the application. If you're running the app, check your taskbar—you should see the **Kestrel console window** where logs are being displayed.



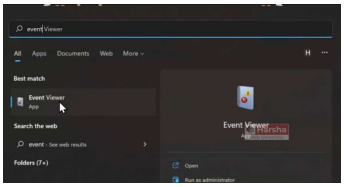
Got it! Here's your refined version with spaces trimmed and spelling fixed while keeping the sentence as it is:

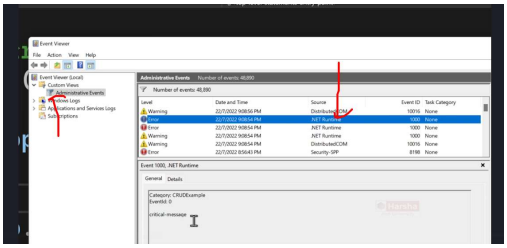
Okay, in the same way, you can see the same type of logs in the **Debug** window as well.
While running the application, come back to **Visual Studio**.
Here, you can find the **Output** window and select **Debug** from the dropdown menu.
Now, you can see all the log messages for **Debug**—this is called a **Debug Provider**.
Actually, a **logging provider** stores or displays the log messages.
Scroll to the top of the **Output** window and search for an **info message**.
Here, you can see **Debug, Information, Warning, Error,** and **Critical** logs—all displayed in the Debug window.
So, when you write a log message, by default, it logs messages to both **Console** and **Debug** windows.





There is one more place where you can find the log messages that is event log in Windows this feature is only for Windows operating system you can go to the start menu and search for Event Viewer this is the application.

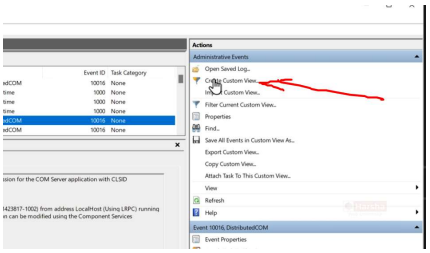


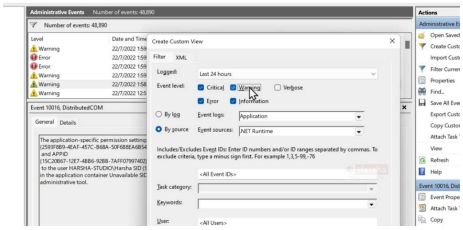


This is the application, so open up the same event viewer here. Okay, this is called event viewer, which can also be opened with control panel. Here, under custom views, you can find administrative views. Here, you can see all our log messages. Check for the log messages with Source equal to dot net 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.

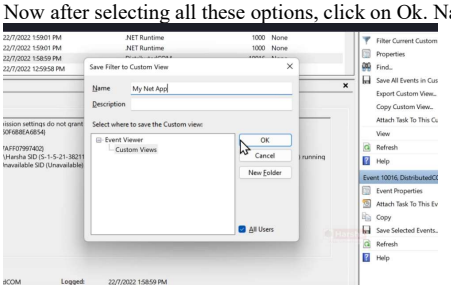
but you can find some other logs which
are not generated by as
these are automatically generated by the
system for a different reason for
different applications as well

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.

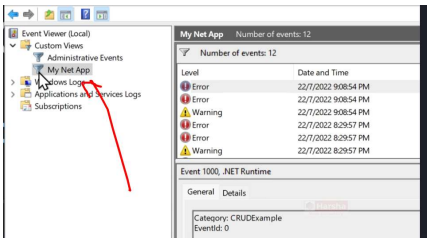


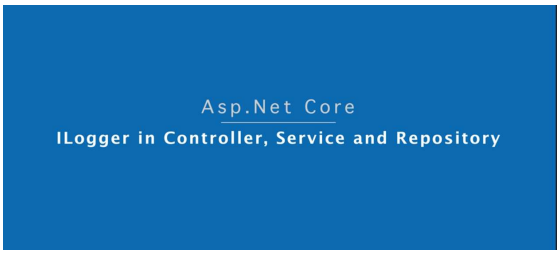


Now after selecting all these options, click on Ok. Name the view for example 'my dot net app'

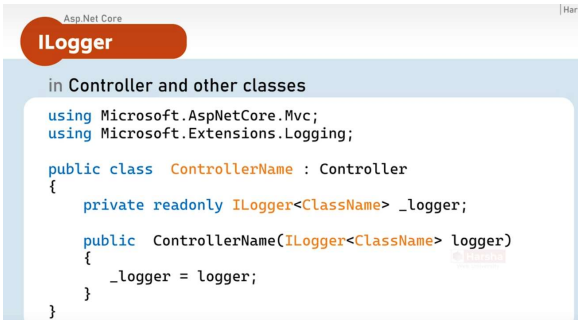


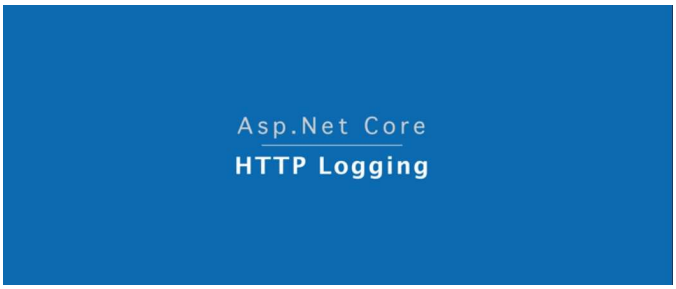
so once you click on the same you can
see all the log levels
that is error warning that are generated
by our code through the program.cs file



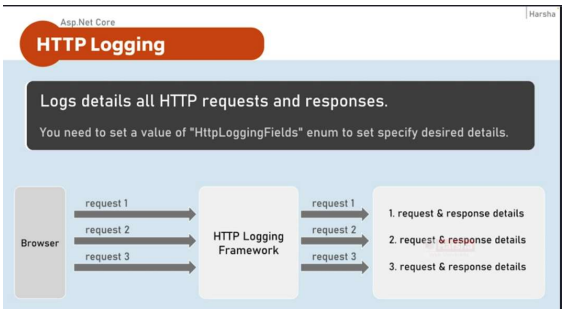


How do you write actual logs? In every layer of the application, such as controllers, services, repositories, you have to leave some log messages which are useful for debugging as well as for tracking the execution sequence of the application. In order to do so, we have to inject a service of ILogger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



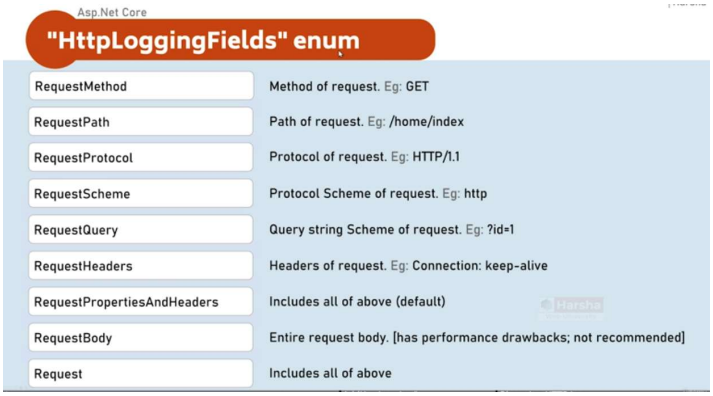


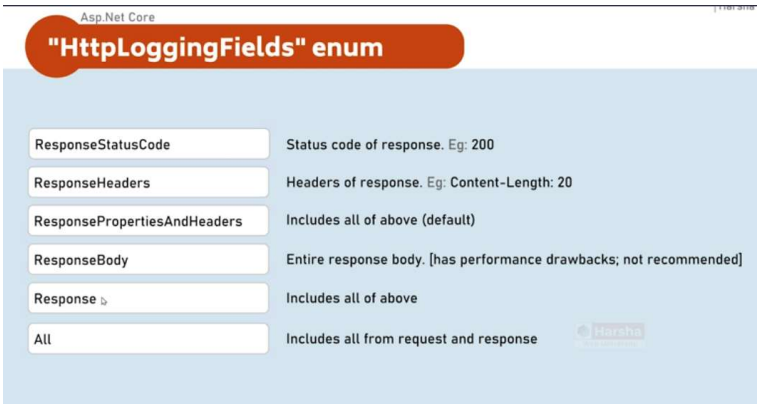
How do you track the incoming requests and corresponding HTTP responses? You can enable HTTP logging in your ASP.NET Core application so that it logs all the requests and corresponding response details in the log. HTTP logging is the process of recording the details of HTTP requests and responses as 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.

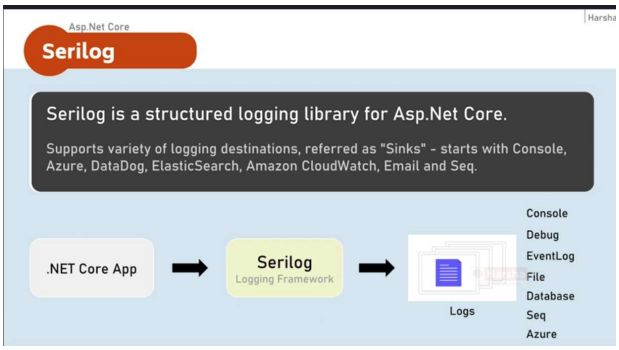


```
Application started. Press Ctrl+C to shut down.
Microsoft.Hosting.Lifetime[0]
Hosting environment: Development
Microsoft.Hosting.Lifetime[0]
Content root path: C:\aspnetcore\CRUDSolution\CRUDExample\
Microsoft.Extensions.Hosting.Internal.Host[2]
Hosting started
Microsoft.AspNetCore.Hosting.Diagnostics[1]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

"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."





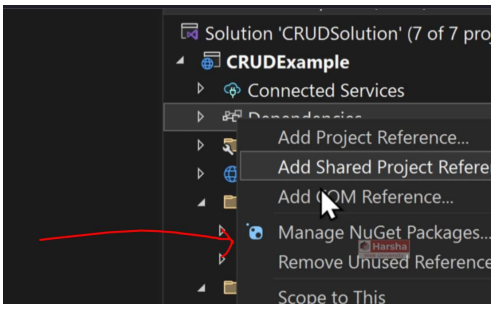


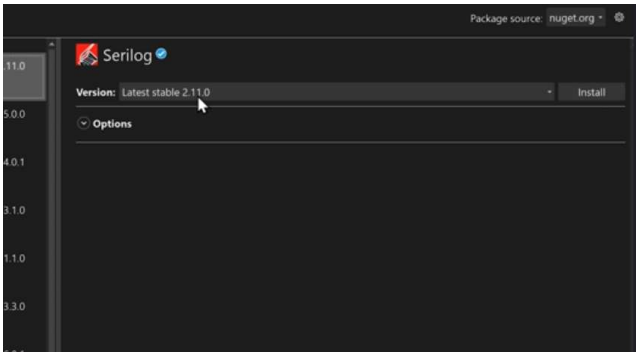
Serilog is one of the most popular logging frameworks in the ASP.NET Core ecosystem. But you might wonder, why use a third-party logging framework when we already have a built-in logging system in .NET Core?

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.

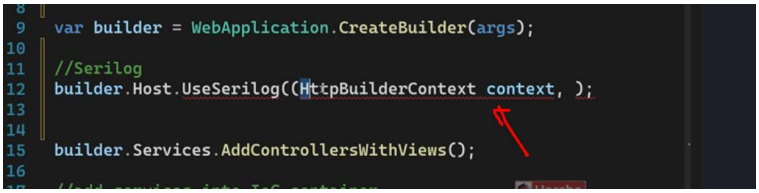
Serilog supports over 30 to 40 different logging destinations, known as 'sinks.' These sinks specify where log messages should be stored. For instance, the 'console' sink outputs log messages to the console window. Similar to the built-in logging system, Serilog also supports debug and event log sinks. 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.

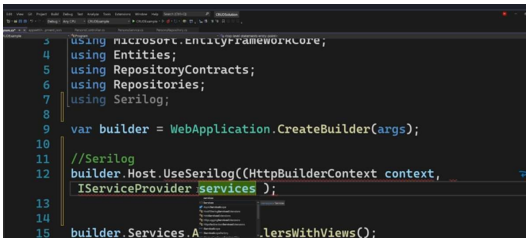




It receives the context of the asp.net core application builder, using which you can access the asp.net core Application configuration or services.



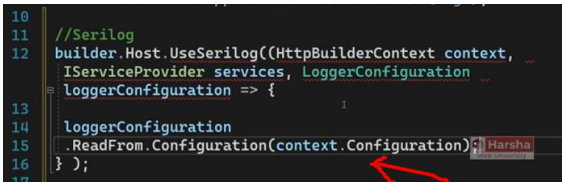
The second one is **IServiceProvider**, this parameter represents the service collection that is built in IOC container in asp.net core application.

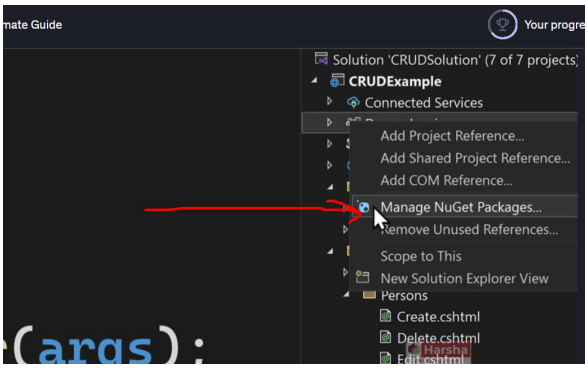


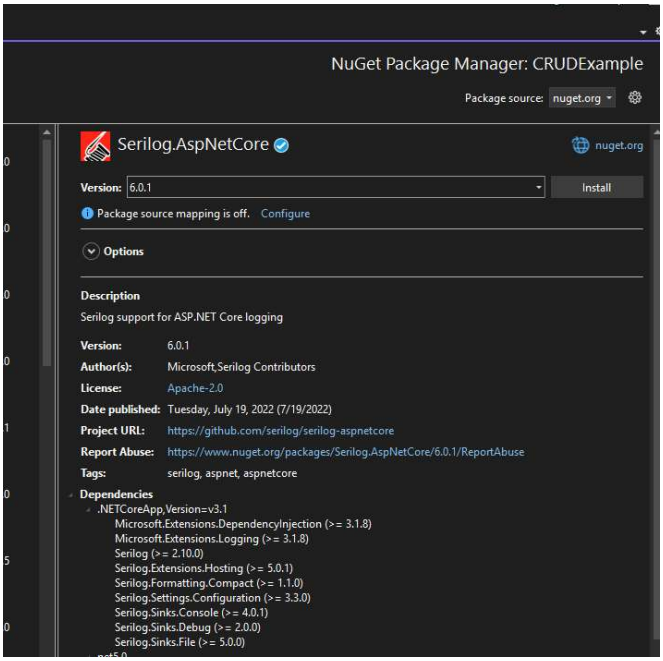
This will represent the configuration of the serilog.

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.

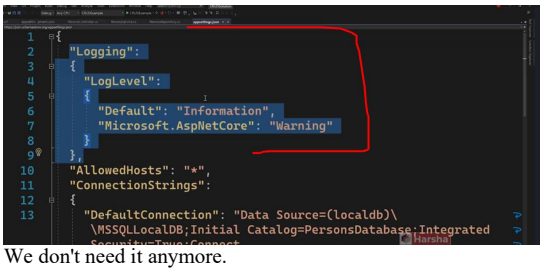
Now the serilog package is able to read the configuration settings of our appSettings.json or appSettings.Development.json based on the environment.







Now open up appSettings.json file



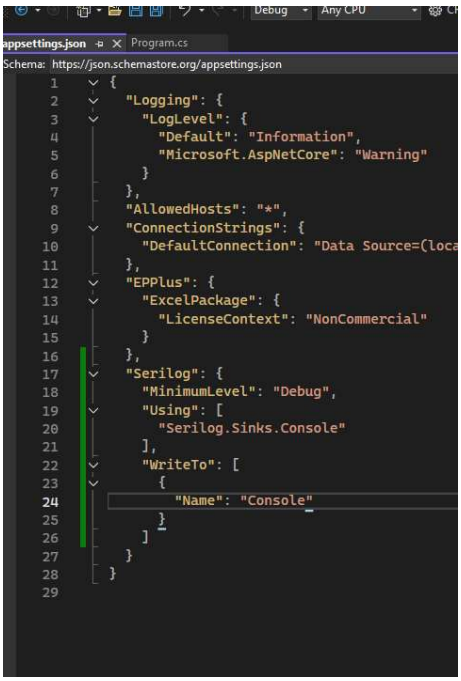
We don't need it anymore.

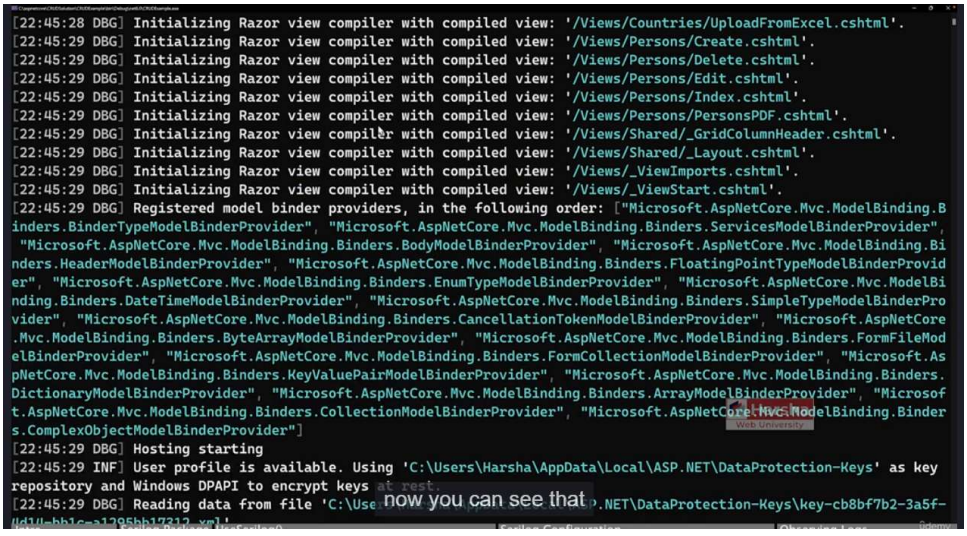
Asp.Net Core

Serilog - Configuration

in appsettings.json

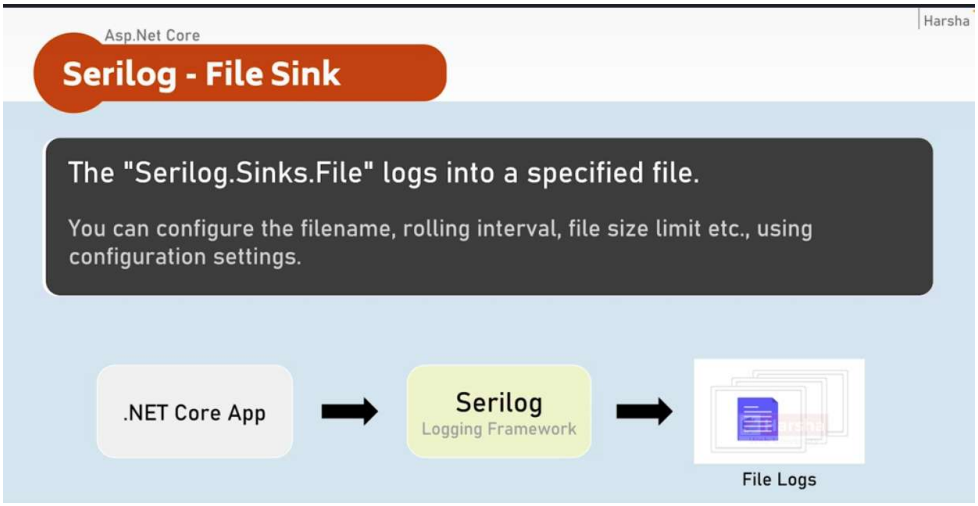
```
{
 "Serilog": {
  "Using": [
      "Serilog.Sinks.YourSinkHere"
  ],
  "MinimumLevel": "Debug | Information | Warning | Error | Critical",
  "WriteTo": [
      {
         "Name": "YourSinkHere",
         "Args": "YourArguments"
      }
   ]
 }
}
```





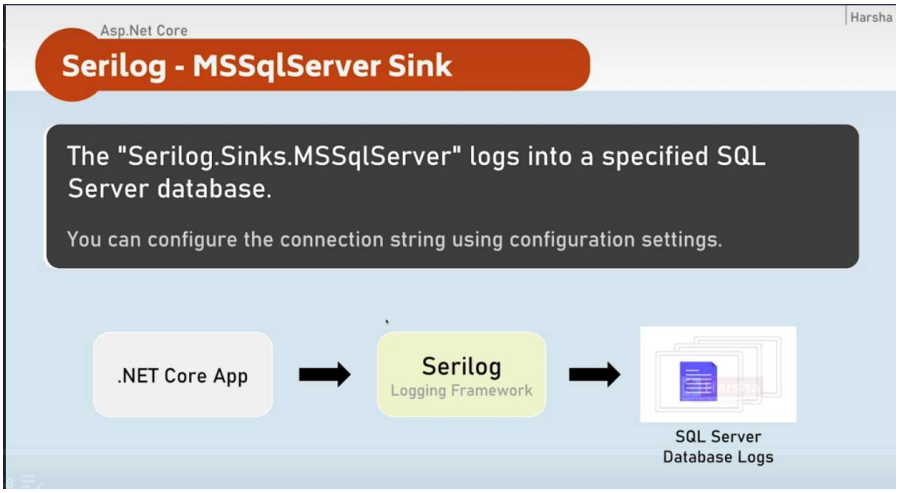
Asp.Net Core

Serilog File Sink

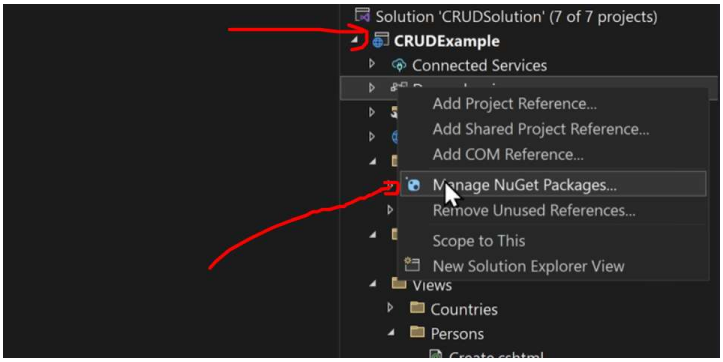


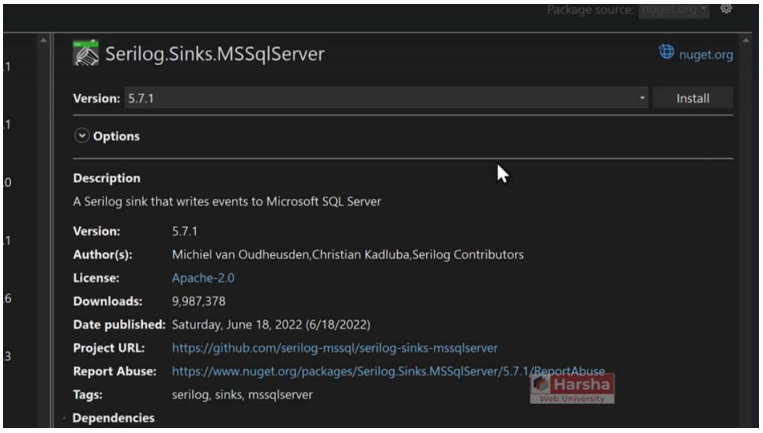
Asp.Net Core

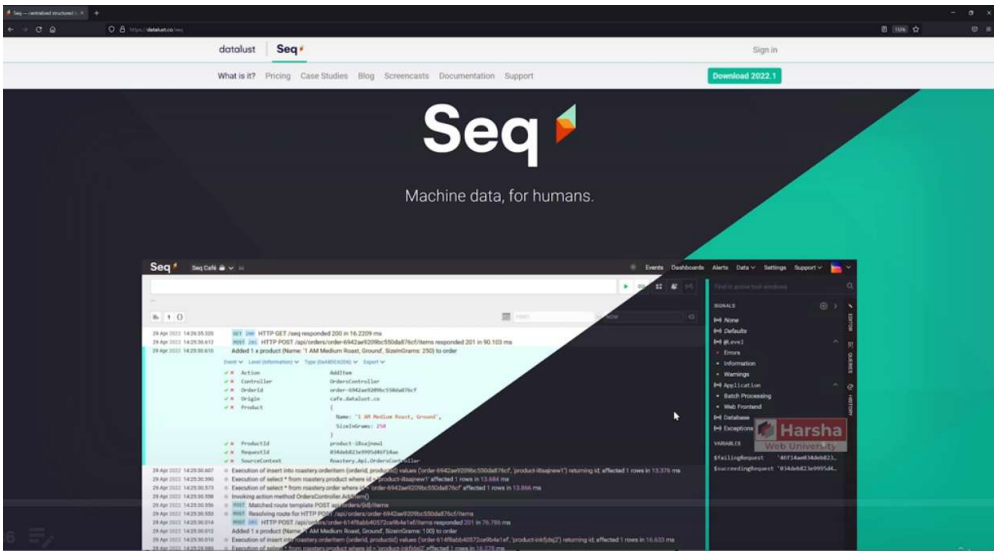
Serilog Database Sink

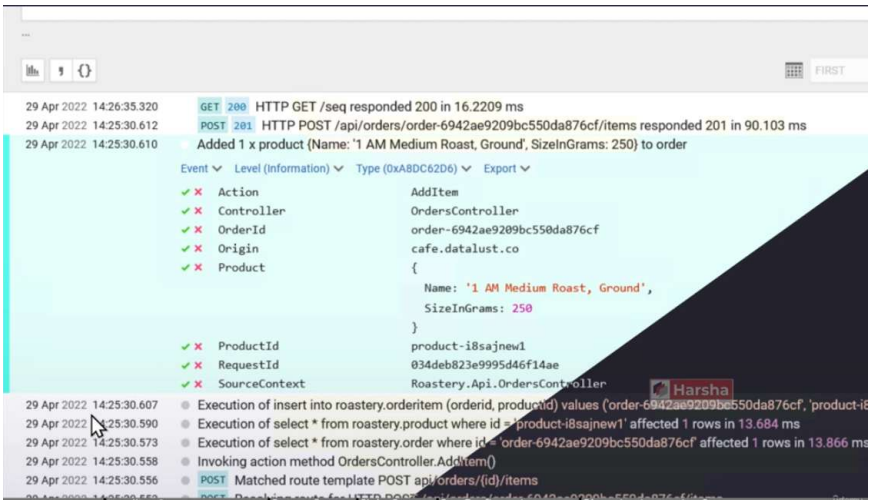


"You can configure Serilog to log directly to an SQL Server database. Serilog not only supports file-based logging but also integrates with various databases such as MS SQL Server, PostgreSQL, MySQL, and Cosmos DB, among others.
In this lesson, we'll focus on configuring MS SQL Server as a sink for Serilog. 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.



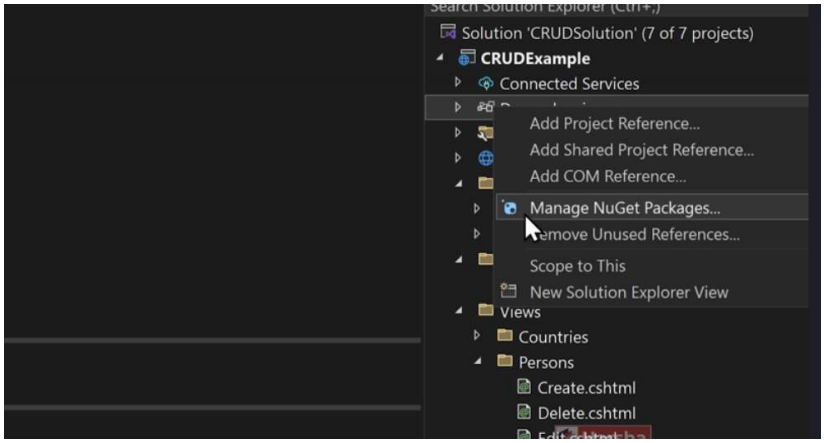


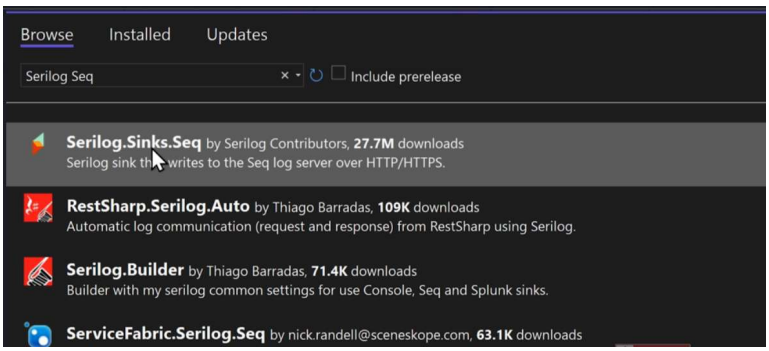


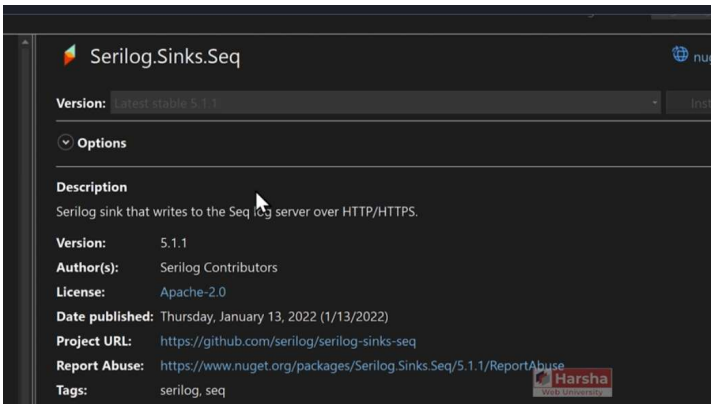


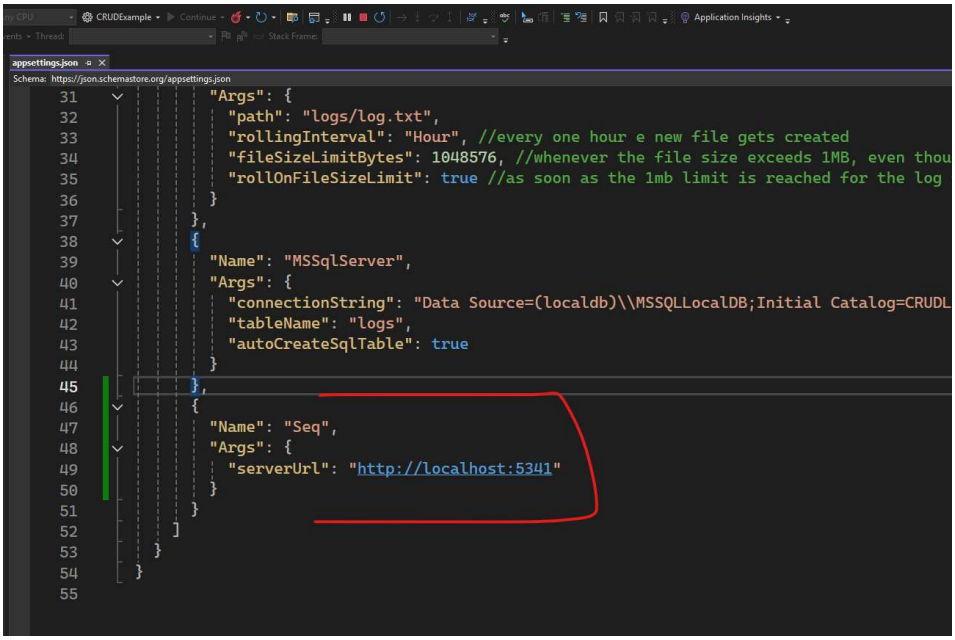
The primary reason for using Seq is that it provides a powerful monitoring tool to track application logs in real time. This allows developers to observe logs being generated for each live request directly from the server.
For example, Seq offers an intuitive interface where you can view logs with timestamps, log levels, and structured messages. By default, Seq supports structured logging, making it easier to filter, search, and analyze logs efficiently.

1. Dowload Seq

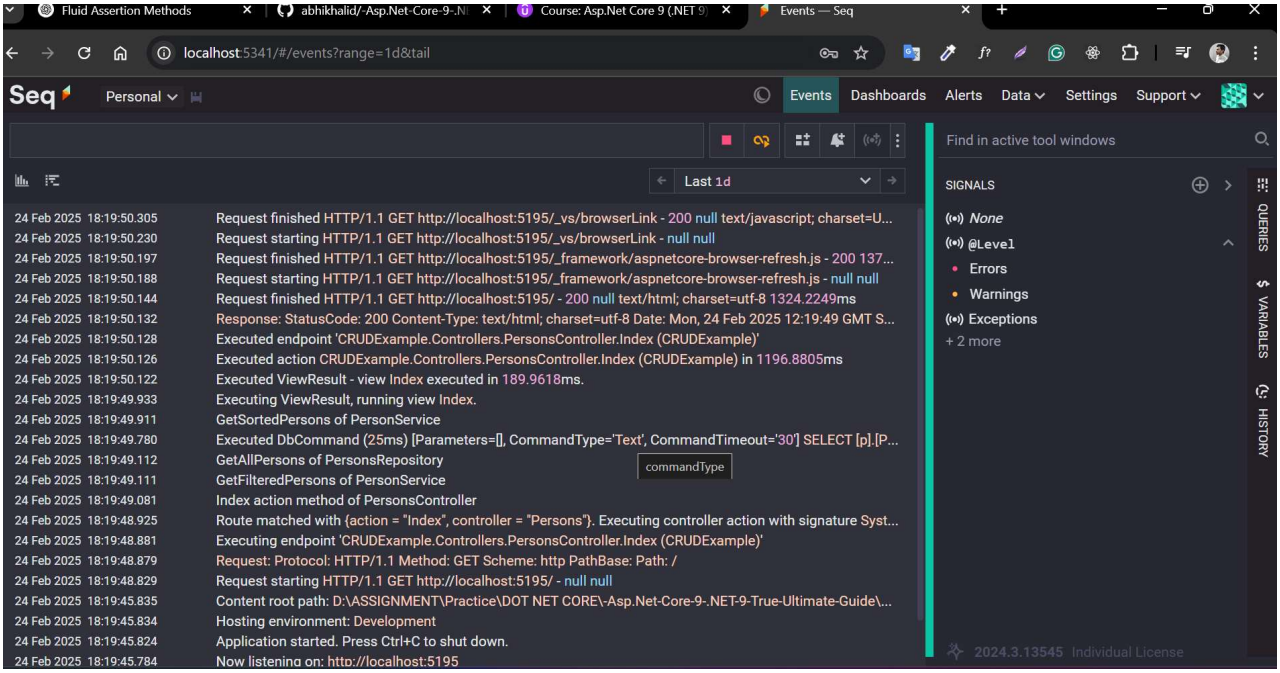
2. 
Solution 'CRUDSolution' (7 of 7 projects)
CRUDExample
Connected Services
Add Project Reference...
Add Shared Project Reference...
Add COM Reference...
Manage NuGet Packages...
Remove Unused References...
Scope to This
New Solution Explorer View
Views
Countries
Persons
Create.cshtml
Delete.cshtml

3. 
Browse
Installed
Updates
Serilog Seq
Include prerelease
Serilog.Sinks.Seq by Serilog Contributors, 27.7M downloads
Serilog sink that writes to the Seq log server over HTTP/HTTPS.
RestSharp.Serilog.Auto by Thiago Barradas, 109K downloads
Automatic log communication (request and response) from RestSharp using Serilog.
Serilog.Builder by Thiago Barradas, 71.4K downloads
Builder with my serilog common settings for use Console, Seq and Splunk sinks.
ServiceFabric.Serilog.Seq by nick.randell@sceneskope.com, 63.1K downloads

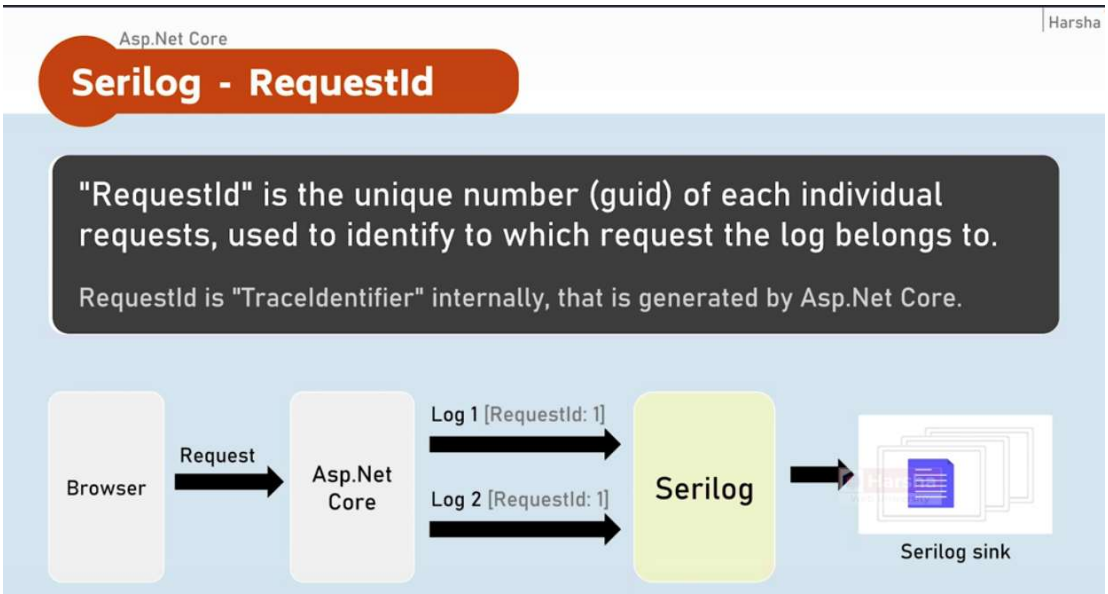

Serilog.Sinks.Seq
Version:
Options
Description
Serilog sink that writes to the Seq log server over HTTP/HTTPS.
Version: 5.1.1
Author(s): Serilog Contributors
License: Apache-2.0
Date published: Thursday, January 13, 2022 (1/13/2022)
Project URL: https://github.com/serilog/serilog-sinks-seq
Report Abuse: https://www.nuget.org/packages/Serilog.Sinks.Seq/5.1.1/ReportAbuse
Tags: serilog, seq
Harsha

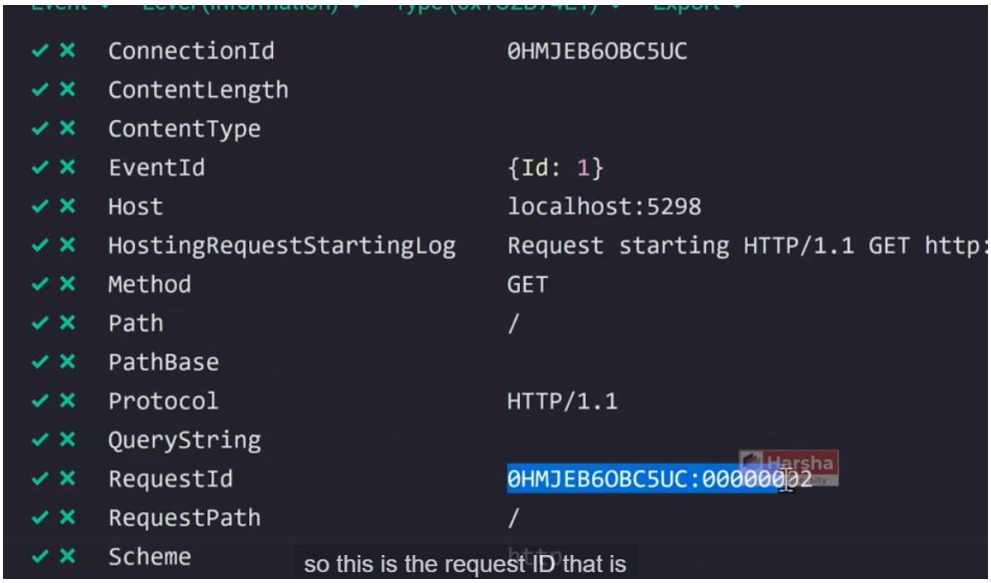


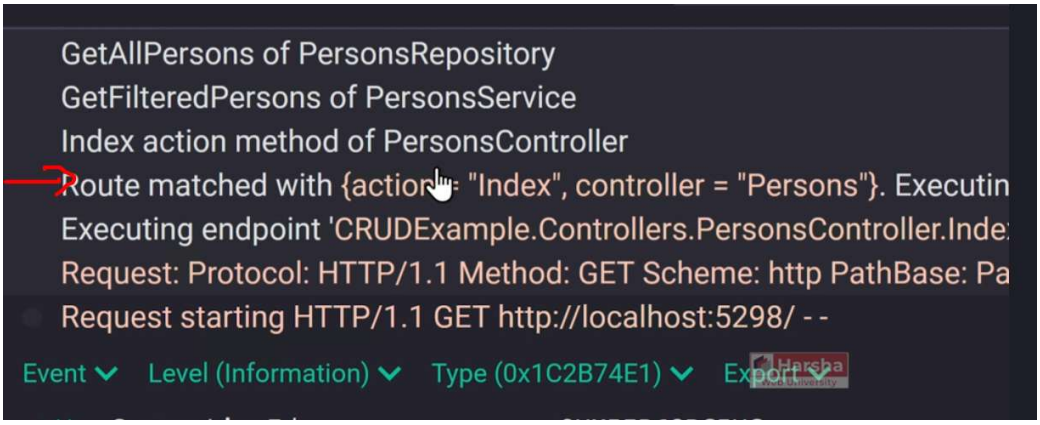
```
appsettings.json
Schema: https://json.schemastore.org/appsettings.json
31          "Args": {
32            "path": "logs/log.txt",
33            "rollingInterval": "Hour", //every one hour e new file gets created
34            "fileSizeLimitBytes": 1048576, //whenever the file size exceeds 1MB, even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



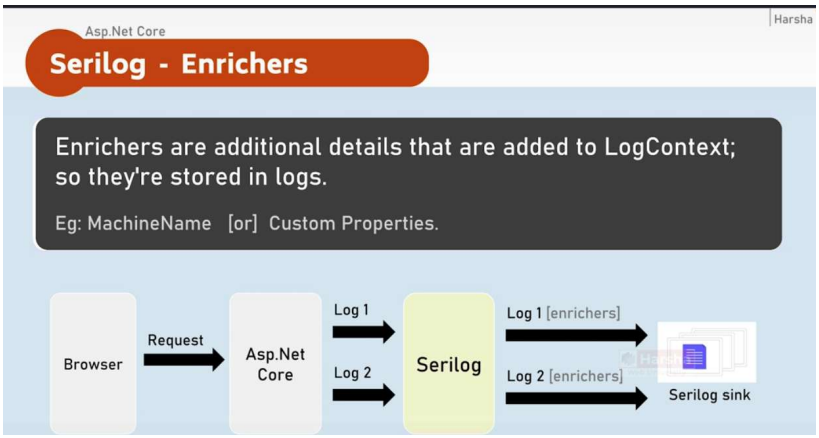
request ID is the unique number that is generated automatically for each request so that you can write all the logs of the request with reference to the a request ID
let me show that




ConnectionId 0HMJEB6OBC5UC
ContentLength
ContentType
EventId {Id: 1}
Host localhost:5298
HostingRequestStartingLog Request starting HTTP/1.1 GET http:
Method GET
Path /
PathBase
Protocol HTTP/1.1
QueryString
RequestId 0HMJEB6OBC5UC:00000002
RequestPath /
Scheme
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"}. Executin
Executing endpoint 'CRUDExample.Controllers.PersonsController.Inde
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 CRUDExample.Controllers.PersonsC
AssemblyName CRUDExample
ConnectionId 0HMJEB6OBC5UC
Controller CRUDExample.Controllers.PersonsC
EventId {Id: 3, Name: 'ControllerActionE
MethodInfo System.Threading.Tasks.Task`1[Mi
lt] Index(System.String, System.
ServiceContracts.Enums.SortOrder
RequestId 0HMJEB6OBC5UC:00000002
RequestPath /



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**.

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:

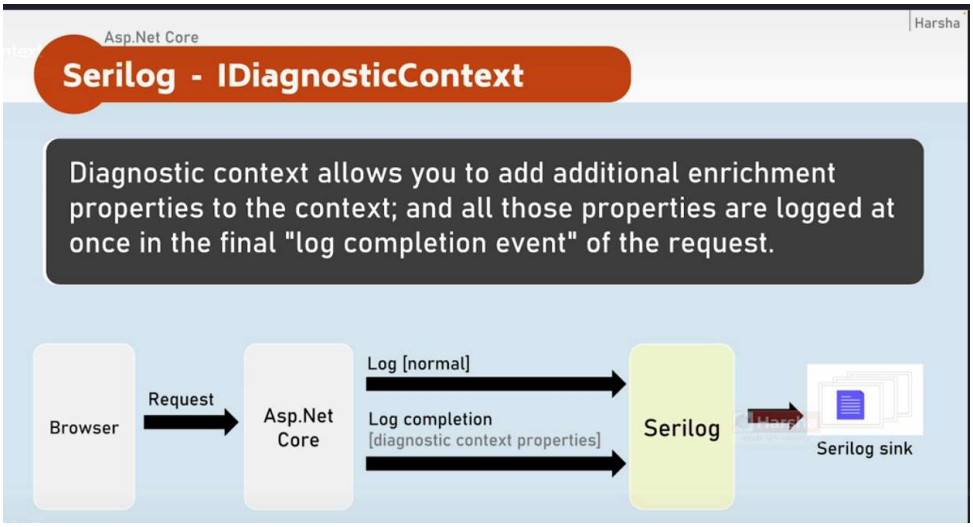
- Adds properties to **every log entry** within the request.
- Useful for tracking values across multiple logs.

2. Diagnostic Context:

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

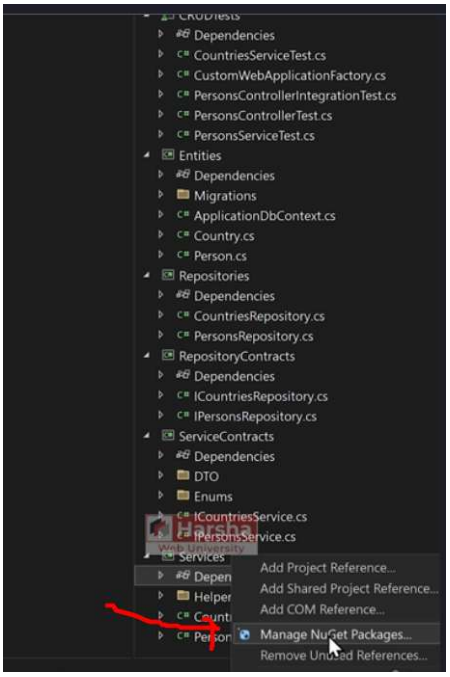
- Ideal for capturing **summary details** about the request.
- Properties can be added dynamically at any layer (Controller, Service, Repository, etc.).
- Each request gets its **own diagnostic context**, identified by a **Request ID**.

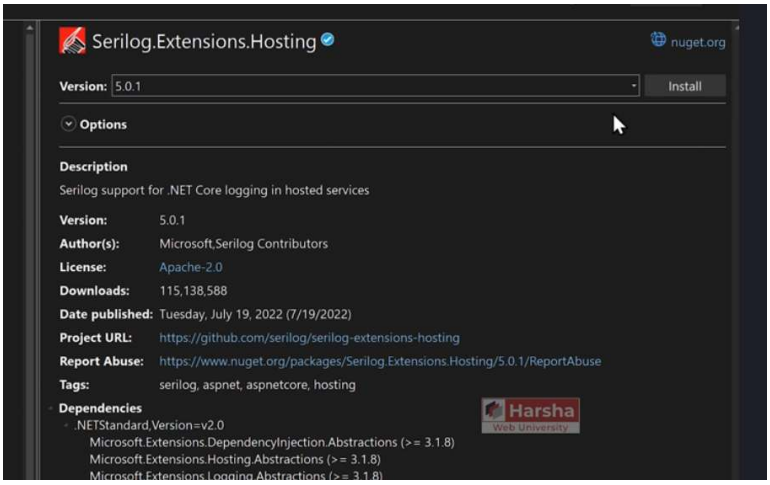
Example Use Case:
Imagine an API request that generates **10 logs** during processing.
. If you use **Log Context**, all 10 logs will include the same extra properties.
. If you use **Diagnostic Context**, only the **last log (completion log)** will include those properties.
This approach helps avoid redundant logging while still capturing crucial information at the **end of the request lifecycle**.

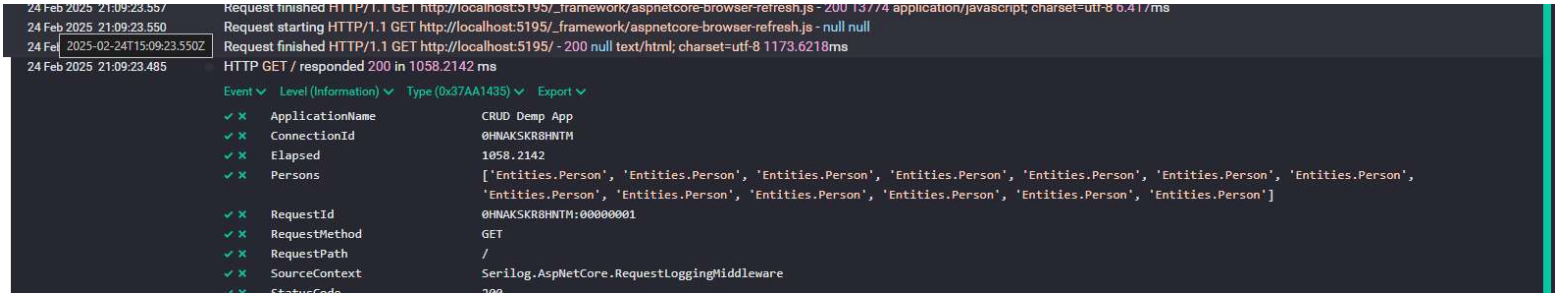


I would like to see the list of persons in the log.

```
7 references
public async Task<List<PersonResponse>> GetAllPersons()
{
 _logger.LogInformation("GetAllPersons of PersonsService");

 var persons = await _personsRepository.GetAllPersons();

 return persons
   .Select(temp => temp.ToPersonResponse()).ToList();
}
```


Dependencies
CountriesServiceTest.cs
CustomWebApplicationFactory.cs
PersonsControllerIntegrationTest.cs
PersonsControllerTest.cs
PersonsServiceTest.cs
Entities
Dependencies
Migrations
ApplicationDbContext.cs
Country.cs
Person.cs
Repositories
Dependencies
CountriesRepository.cs
PersonsRepository.cs
RepositoryContracts
Dependencies
ICountriesRepository.cs
IPersonsRepository.cs
ServiceContracts
Dependencies
DTO
Enums
ICountriesService.cs
IPersonsService.cs
Services
Depen
Helper
Counti
Person
Add Project Reference...
Add Shared Project Reference...
Add COM Reference...
Manage NuGet Packages...
Remove Unused References...


Serilog.Extensions.Hosting
nuget.org
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)
Harsha
Web University


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 6.417ms
24 Feb 2025 21:09:23.550
Request starting HTTP/1.1 GET http://localhost:5195/_framework/aspnetcore-browser-refresh.js - null null
24 Fe
2025-02-24T15:09:23.550Z
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
Event
Level (Information)
Type (0x37AA1435)
Export
ApplicationName CRUD Demp App
ConnectionId 0HNAKSKR8HNTM
Elapsed 1058.2142
Persons ['Entities.Person', 'Entities.Person', 'Entities.Person', 'Entities.Person', 'Entities.Person', 'Entities.Person', 'Entities.Person', 'Entities.Person', 'Entities.Person', 'Entities.Person', 'Entities.Person', 'Entities.Person', 'Entities.Person']
RequestId 0HNAKSKR8HNTM:00000001
RequestMethod GET
RequestPath /
SourceContext Serilog.AspNetCore.RequestLoggingMiddleware

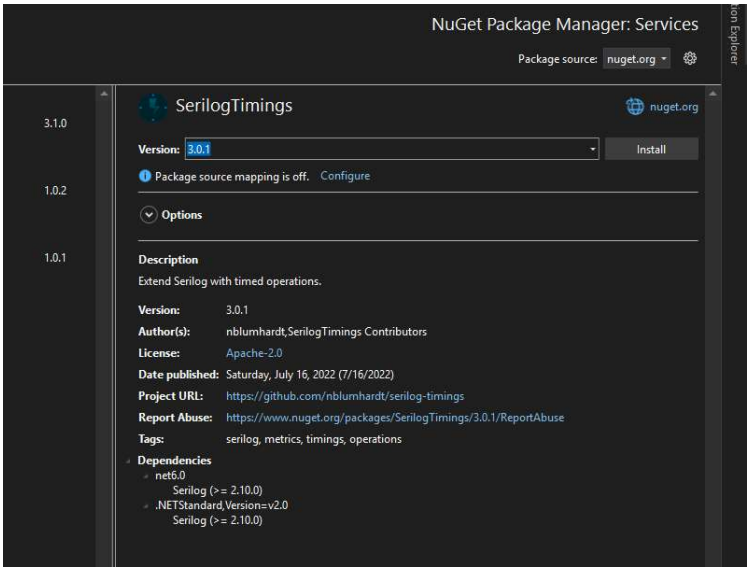
The **Serilog Timings** is another package that works on top of **Serilog**. It is one of the best ways to record the time taken for the execution of a particular block of code and add the same into the logs. For example, there is a method that contains a massive amount of code, or a method that connects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

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 nuget packages'


By switch
Name) =>
FilteredPersons(tem
archString)),
=>
Dependencies
Migrations
ApplicationDbContext.cs
Country.cs
Person.cs
Repositories
Dependencies
CountriesRepository.cs
PersonsRepository.cs
RepositoryContracts
Dependencies
ICountriesRepository.cs
IPersonsRepository.cs
ServiceContracts
Dependencies
DTO
Enums
ICountriesService.cs
IPersonsService.cs
Harsha
Add Project Reference...
Add Shared Project Reference...
Add COM Reference...
Manage NuGet Packages...


NuGet Package Manager: Services
Package source: nuget.org
3.1.0
1.0.2
1.0.1
SerilogTimings
nuget.org
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
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=[], CommandType='Text', CommandTimeout='30'] SELECT [p].[PersonID], [p].[Address], [p].[CountryID]