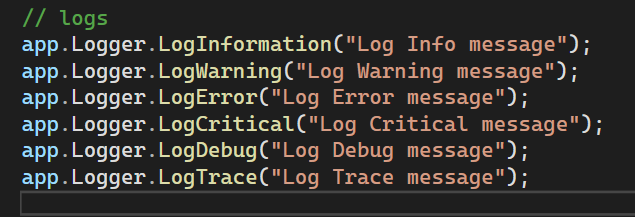
**Logging in Asp.Net**

* Is the process of recording run-time actions as they happen in real-time
* Helps to understand failures and bottlenecks of the application
* Logging can be done using
  + Console -> changes are not persisted for further reviews
  + Debug
  + Event Log
  + File
  + Database
* ILogger is interface that provides implementation of logs

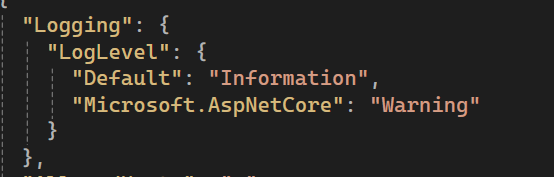
1. Debug => ILogger.LogDebug(“msg”)
2. Information = > ILogger.LogInformation(“msg”)
3. Warning => ILogger.LogWarning(“msg”)
4. Error => ILogger.LogError(“msg”)
5. Critical => ILogger.LogCritical(“msg”)
6. Trace => ILogger.Trace(“msg”)



\*\* These logs are displayed in the Kestrel window be default debug is not visible in kestrel window we need to configure in the appsetting.json file

**Logging Configuration**

* By default, debug log is not visible to kestrel window. We need to enable it in appsettings.json

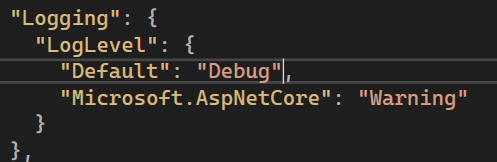


By default it was set to till information

Order: Debug | Information | Warning | Error | Critical

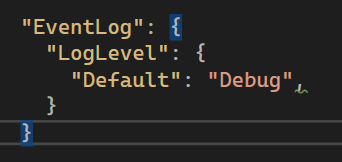
If set to Warning it logs only Error and Critical (>=)

* We need to set to default to debug to try all in both appsettings.json and appsettings.development.json

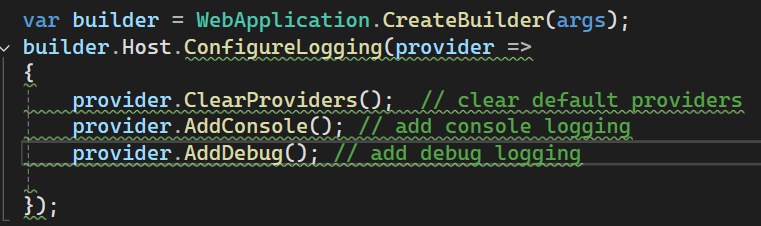


**Log Providers**

* Logging providers specifies where to store/display logs
* Built-in loggers support only in Console, Debug, EventLog and EventSource and by default when application is running logs are displayed in both console, debug and Event Log(minimum is Warning) in Windows Os

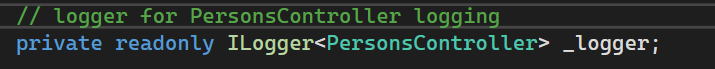
set for overriding in event log

* Have no support for File and Database
* We can also override default logging and can add only specific loggings



**ILogger**

* At each stage(level) of application we need to log the appropriate messages
* We can use ILogger to inject logger into current class and provide suitable messages there and inject in its class constructor

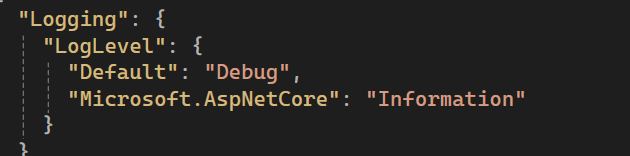
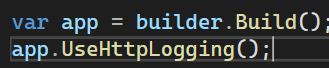


* Register the ILogger as service in program file



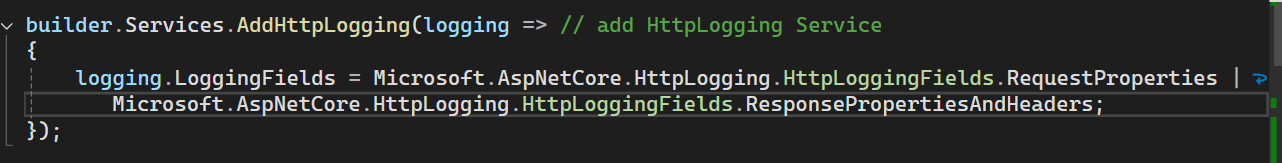
**Http Logging**

* Is the process of recording request and response details when communication is happening
* Register HttpLogging as service in builder
* Register the HttpLogging middleware in program.cs file
* And in the settings.json(both) change Microsft.AspNetLogLevel to Information



* It logs request url, headers, route but doesnot logs Request body and Response body automatically we need to enable in options

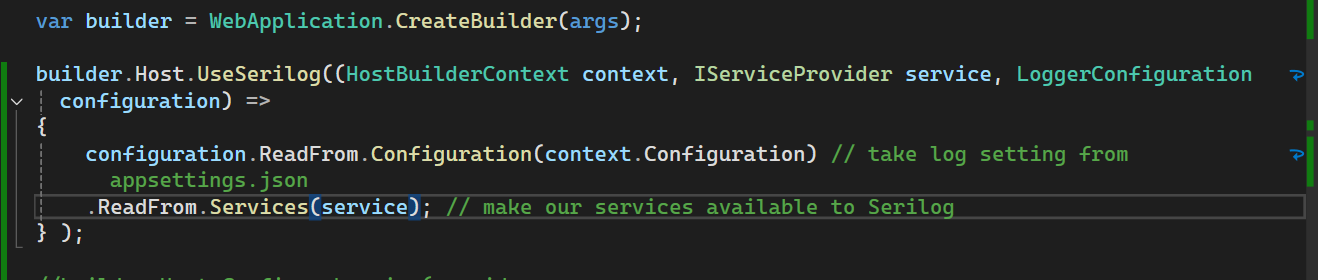
**Http Logging Options**

* You can set to log which properties of Http Request and Response (like Method, Request Body etc)
* 

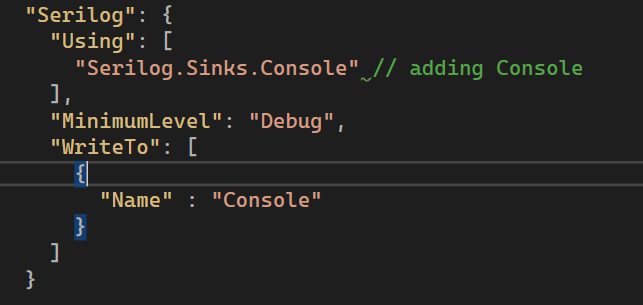
**SeriLogging**

* It is a 3rd party framework that supports logging for more than 40 destinations (including files, debug etc..) these are called sinks
* Install 2 packages (Serilog and Serilog.AspNetCore) into Main project

1. Configure Serilog in Program.cs



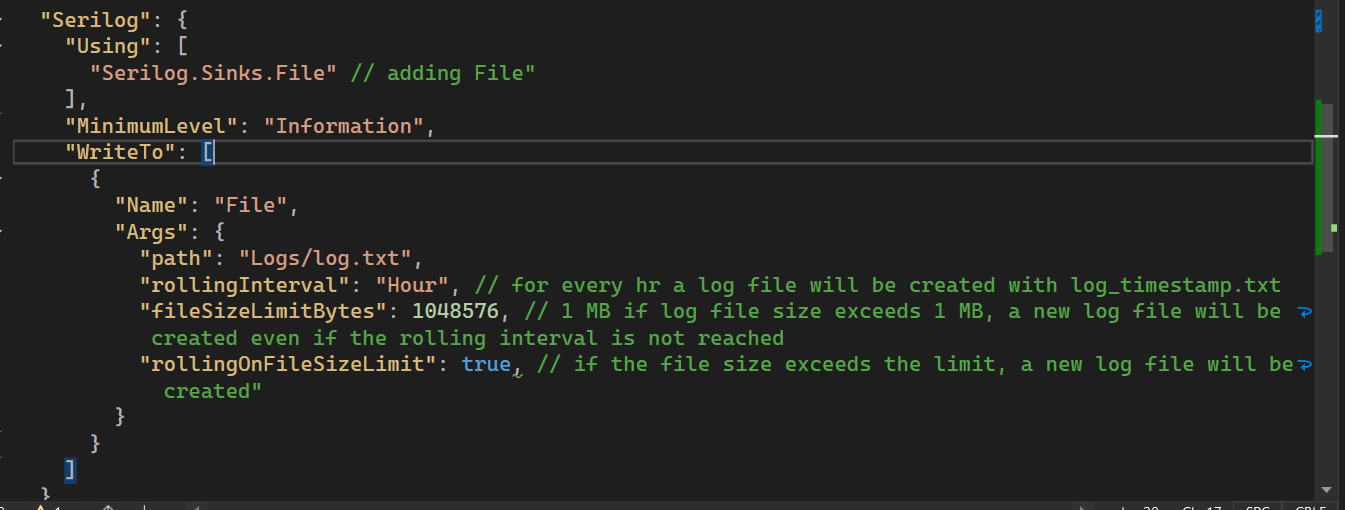
1. In appsettings.json file add SeriLog option to configure. Here you need to mention:
   1. Type of SeriLog sinks as array
   2. Minimum Level of log
   3. Write to name and Arguments specific to a sink(additional)



1. Now we can use ILogger as above.

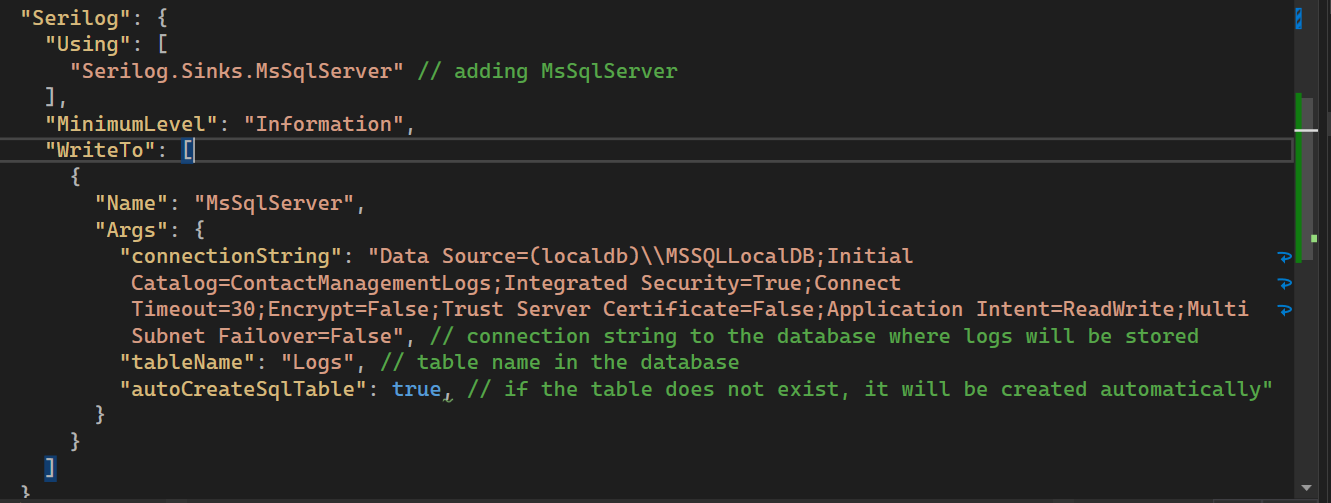
**Writing SeriLog to a File – Persisting Logs permanently**

* To write into file, you need to add Serilog.Sinks.File in Using section of settings
* You can configure the filename, rolling interval, file size limit etc



**SeriLog Database sink:**

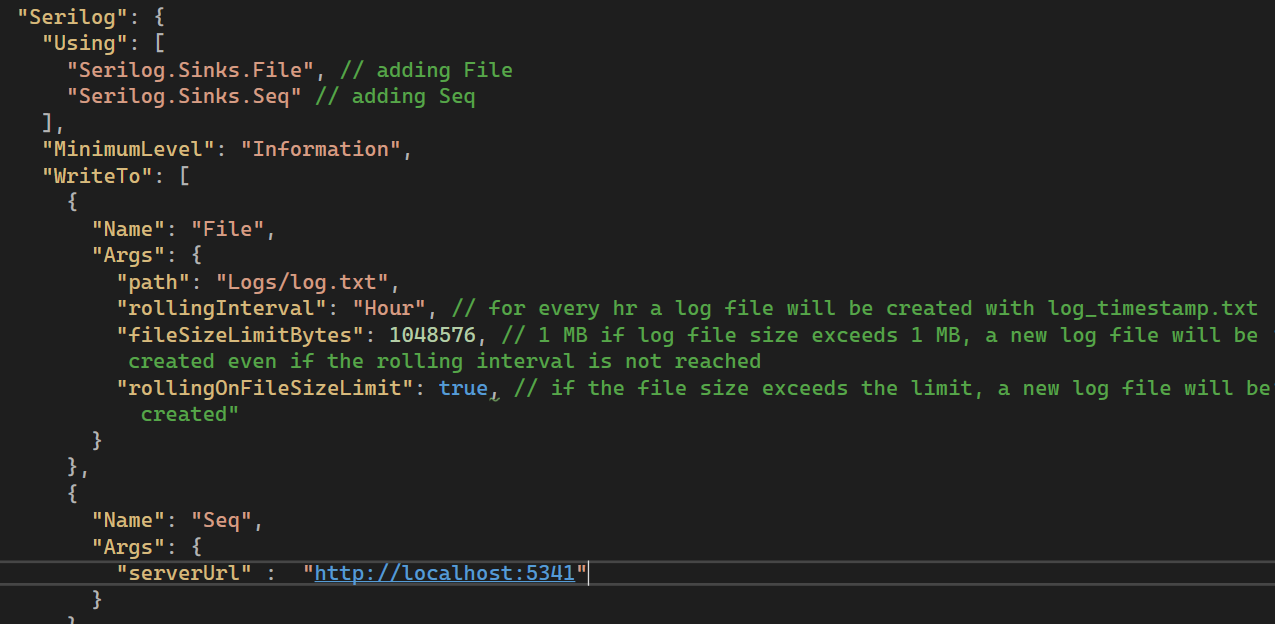
* Install Serilog, Serilog.AspNetCore and SerLog.Sinks.MsSql for storing logs in ms sql server
* Similar to File sink add settings of database in appsettings Serilog
* We need to create a new separate database to store all logs (manually)
* SeriLog can create table automatically when you mention it in logs

****

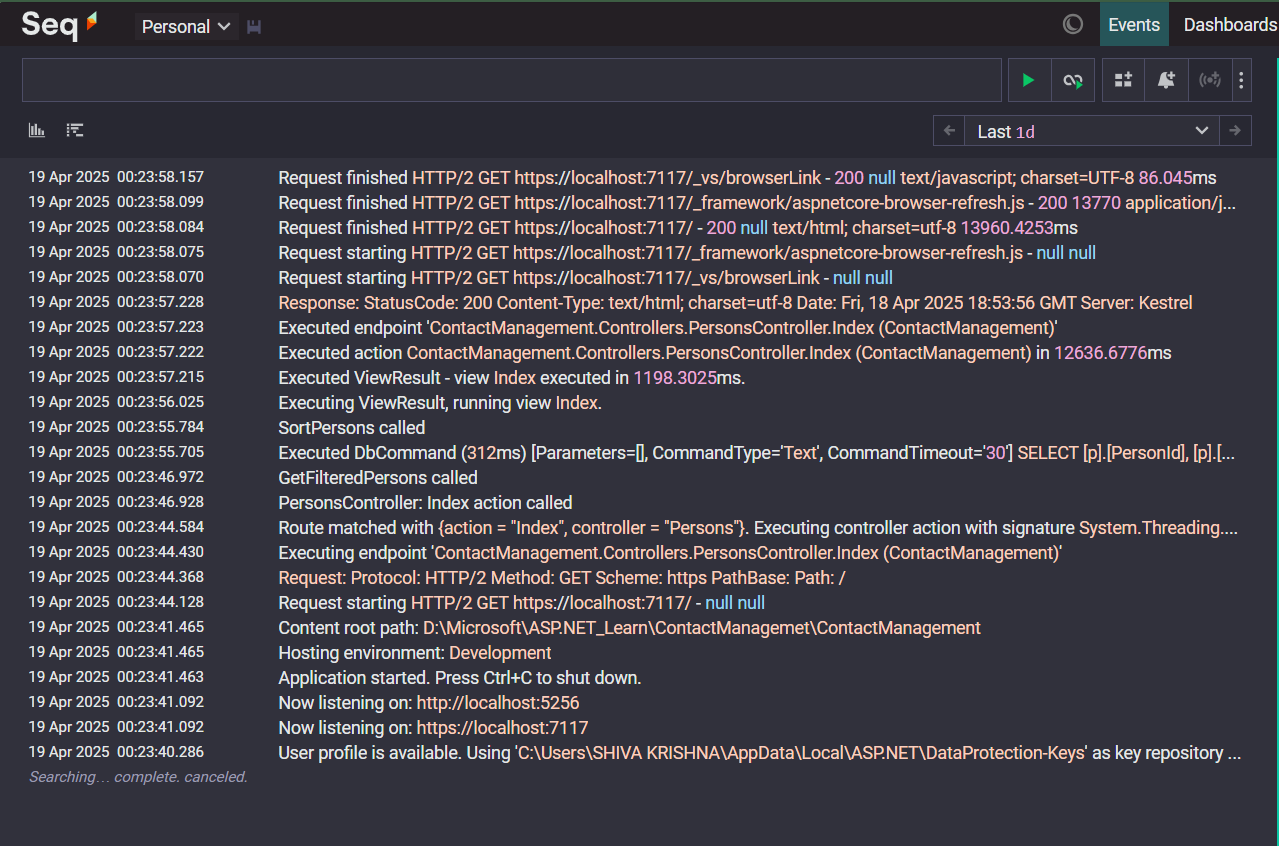
**\*\* here in image “Name”: “MSSqlServer” case sensitive\*\***

**SeriLog Seq – Monitoring Tool for Logs**

* Install the Seq software in to the system (datalust.co.Seq) [Link](https://datalust.co/download)
* Install it and set password (username as admin)
* Seq will run separately on localhost:5341 open it and login and enable the Tail
* Install SeriLogs, SeriLogs.AspNet, SeriLogs.Seq packages into your main project
* Configure the settings in appsettings.json file



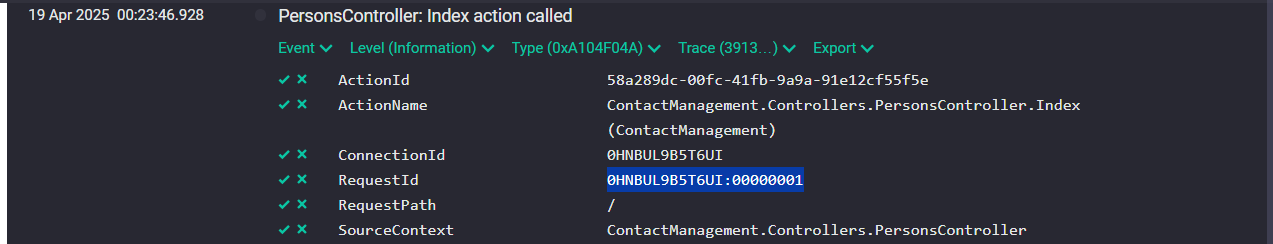
* Now logs will be displayed in Seq in desc order of timestamp



* Advantage is that developer get more details when he clicks in that particular request

**SeriLog – RequestId**

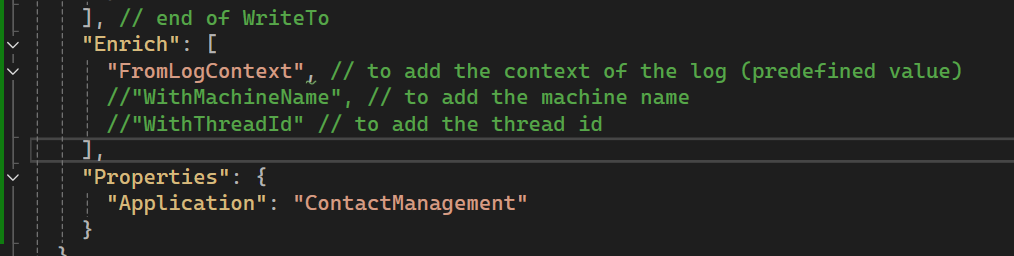
* Request Id is the unique number(guid) for each request generated by aspnet automatically that identifies which request log belongs to
* It is a TraceIdentifier internally(in aspNet) generated on each new request

****

* Advantage is that any future request that depends on current request will carry this request Id as reference (from where I was called)

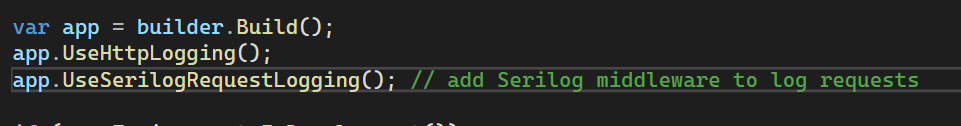
**SeriLog Enrichers**

* Are additional details that are added to LogContext, like MachineName, CustomProperties etc
* To add Custom properties in our appsettings.json serilog configs add new property(after Write To array) “Enrich – array” and “Properties – Key Value pairs”

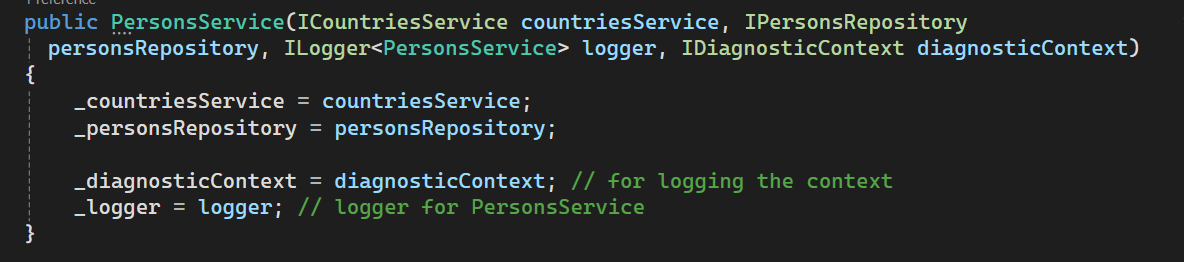


**SeriLog IDiagnostic context – for additional dynamic properties**

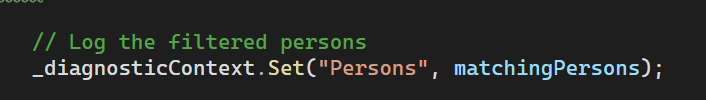
* Say you want to log some properties only when some condition met in such cases we use IDiagnostic Context
* It allows you to add additional enrichment properties to context and all these properties are logged at once in the final log completion event of the request
* Ie, in normal if request has 10 logs generally we see 10 logs but with IDiagnostic we see log at end of request
* Allows to add dynamic values at any layer of applications
* Install SeriLog.Extensions.Hosting package into project in which you are using IDiagnostic Context
* In main project add app.UseSerialLoggingRequest()



* Create a field and inject to constructor



* Now use Set(key,value) method to log



**SeriLog Timing:**

* Records the execution timing of your code block
* In the project that want to record time, install SeriLog Timings package
* Using(Operation.Time(“some text”)){

/// code

}’