Instructor example

Web University by Harsha Vardhan

Explain how logging works in Asp.Net Core?

Logging is the process of recording events in software as they happen in real-time, along with other information such as the infrastructure details, time taken to execute, etc. Logging is an essential part of any software application. Having logs is crucial, especially when things go wrong. Logs help you understand the failures or performance bottlenecks and fix the problems.

Logs are typically written to a database, console, or file, depending on the severity of the application and convenience. Although it's possible to record any data in the logs, you write informational messages and error messages in general. The informational messages capture standard events as they happen, e.g. a method call, user authentication, product checkout, etc. The error messages capture the errors and provide all the data that might help you debug the program.

Configuring logging providers in Asp.Net Core:

```
builder.Host.ConfigureLogging(logging =>
{
   logging.ClearProviders();
   logging.AddConsole();
   logging.AddDebug();
   logging.AddEventLog();
});
```

When a log event generated using ILogger, the same log will be added to all logging providers added in ConfigureLogging() method.

What is Serilog and why to use it?

Serilog is the reference example of a Structured Logging system - one that doesn't just create messages, it easily captures key attributes and data about the context the message happened in. While this is possible with MEL via Logging Scopes (and desirable!), Serilog does it automatically as a natural outcome of composing log messages from format strings, a neat trick!

Advantages:

- Structured logging
- Enrichment
- Great documentation and community
- Support for vast number of Sinks such as Azure Analytics, Amazon CloudWatch, elmah.io, Email, Seq, SqLite, SQLServer etc.

What is IDiagnosticContext in Serilog and how to use it?

The diagnostic context is provides an execution context (similar to LogContext) with the advantage that it can be enriched throughout its lifetime. The request logging middleware then uses this to enrich the final “log completion event”. This allows us to collapse many different log operations into a single log entry, containing information from many points in the request pipeline.

In Controller / Service:

```
private readonly IDiagnosticContext _diagnosticContext;

public ClassName(IDiagnosticContext diagnosticContext)
{
  _diagnosticContext = diagnosticContext;
}

public return_type method()
{
  // The request completion event will carry this property
  _diagnosticContext.Set("key",”value”);
}
```