Security Subject - Logging Onderzoek

Wat is het nut van loggen in mijn applicatie?

Introductie:

Ik wil in dit onderzoek graag uitzoeken waarom ik zou moeten loggen in mijn applicatie. Wat kan logging voor je betekenen in je applicatie? Heeft het ook zwakke punten? Ik ga hieruit kunnen halen of logging voor mij nodig is, of niet.

Ik gebruik voor elk van mijn onderzoeken het welbekende dot-framework. Ik zal bronnen aangeven bij iedere vraag die ik stel en daarnaast specifieke bronnen gelijk vermelden onder o.a screenshots/quotes. https://ictresearchmethods.nl/The_DOT_Framework.

Wat is logging?

Met logging kun je logs opslaan in bestanden, hoewel logs die verschijnen in de console, alleen worden weergegeven op de console. Logs zijn bepaalde handelingen die worden uitgevoerd als je een applicatie runned. Vandaar dat je tijdens de runtime van de applicatie in de console ook allerlei messages krijgt; logs. Persoonlijk gebruik ik logging in elke try-catch of if-else statement.

Je kunt met behulp van logging allerlei problemen opsporen en zo heb je veel meer controle over je applicatie. Vaak wordt loggen overgeslagen in het software development process totdat het programma crasht. Als het programma crasht kun je debuggen wat je maar wilt, maar met logging kun je zo in de log-files terug zien waar het fout gaat. Het is belangrijk wat voor data je daadwerkelijk logged. Anders kan het voor research teams zeer onoverzichtelijk worden in grotere applicaties. Hoe log je op de juiste manier?

```
try
{
    _log.Debug("About to do something");
    // ...
}
catch (Exception ex)
{
    _log.Error("Doing something failed with", ex);
}
```

br: https://michaelscodingspot.com/logging-in-dotnet/, https://www.loggly.com/ultimate-guide/net-logging-basics/

Hoe moet je juist loggen? Wat moet je doen en wat niet?

Note: Bad practices zijn het tegengestelde van good practices, dus hou de volgende good practices aan bij het loggen:

- **Maak niet je eigen logging systeem.**

Het is natuurlijk mogelijk om een logging systeem te maken maar het is zeer onnodig en onhandig. Er zijn meerdere opties die al volledig werkend en getest zijn waarvan één die al met de .NET SDK meegeleverd word. Andere bekende opties zijn Log4Net, serilog en mijn keuze; NLog. Reden daartoe was dat ik de setup van het systeem gemakkelijk begreep vandaar dat een andere keuze onnodig was.

- **Informatie loggen naar bestanden, ipv de console.**

Het is ongetwijfeld fijn om naar de console te loggen, hoewel het onhandig is. Als je wat vaker debugged en results wilt vergelijken is het het beste om naar bestanden te loggen. Je krijgt dezelfde informatie door, alleen dan opgeslagen en klaar voor vergelijking.

- **Als je informatie logged, geef het een naam; een level.**

Een van de meest handige features van een 3rd party library zoals NLog is het levelen van bepaalde logs. Je kan de volgende levels meegeven (in de juiste volgorde gebaseerd op prioriteit): ALL, DEBUG, INFO, WARN, ERROR, FATAL, OFF. Je kan configuratie files zodanig opstellen dat bepaalde dingen worden genegeerd en sommige juist wel worden aangetoond. In het volgende voorbeeld zijn INFO en DEBUG meegegeven. Hier kan laterna weer eventueel op gefilterd worden.

```
private static readonly ILog log = LogManager.GetLogger(typeof(MyApp));
log.Info("Starting application.");
log.Debug("DoTheThing method returned X");
```

https://raygun.com/blog/c-sharp-logging-best-practices/

- **Structured logging.**

Als er gestructureerde data in de logs zit kan dit goed gefilterd worden door tools als Loggly en Papertrail. Zo kun je door de log files zoeken wanneer het nodig is om bepaalde resultaten erbij te halen om problemen op te lossen. Als voorbeeld:

```
var requestInfo = new { Url = "https://myurl.com/data", Payload = 12 };
_log.Information("Request info is {@RequestInfo}", requestInfo);
```

hoofdstuk br: https://michaelscodingspot.com/logging-in-dotnet/,
https://raygun.com/blog/c-sharp-logging-best-practices/

Nu we weten wat logging is en hebben meegekregen wat er belangrijk is bij het loggen, gaan we kijken welke frameworks er bestaan voor het loggen. Welke zijn handig te implementeren? Is het überhaupt nodig iets te implementeren of levert .NET (hierin zal mijn API worden gemaakt) een eigen logging service?

In de .NET space zijn er voornamelijk 4 frameworks die domineren. Ze zijn vergelijkbaar met elkaar en allemaal gratis. **Dit sluit netjes aan op mijn criteria. Perfect voor een student dus.**

Deze 4 zijn: **log4net, NLog, Serilog, and Microsoft.Extensions.Logging.**
br: https://michaelscodingspot.com/logging-in-dotnet/

Logging frameworks

Logging frameworks leveren meestal het volgende als features:

- **Logging levels** ("Hoe moet je juist loggen? Wat moet je doen en wat niet? > Als je informatie logged, geef het een naam; een level.")

- **Logging targets** ("Hoe moet je juist loggen? Wat moet je doen en wat niet? > informatie loggen naar bestanden, ipv de console.")

- **Structured logging** ("Hoe moet je juist loggen? Wat moet je doen en wat niet? > Structured logging.")

Een aantal logging frameworks in the spotlights:

NLog

NLog is een van de meest populaire en best-performing frameworks. Het is heel erg simpel om NLog op te zetten in .NET. Je kan de Nuget package downloaden en dan in de NLog.config file allerlei **targets** opzetten. Targets zijn de plekken waar je naartoe kan loggen. Zoals de console, mail of gewoon een kladblok. *Wrappers* is hetgeen dat kan aanpassen hoe de target werkt en het gedrag van de target kan verbeteren. Zo is er bijvoorbeeld een AsyncWrapper die asynchroon logs versturen naar de targets

NLog support deze logging levels:

- DEBUG: Additional information about application behavior for cases when that information is necessary to diagnose problems
- INFO: Application events for general purposes
- WARN: Application events that may be an indication of a problem
- ERROR: Typically logged in the `catch` block a try/catch block, includes the exception and contextual data
- FATAL: A critical error that results in the termination of an application
- TRACE: Used to mark the entry and exit of functions, for purposes of performance profiling

br:https://raygun.com/blog/c-sharp-logging-best-practices/#:~:text=NLog%20supports%20the%20following%20logging%20levels%3A

NLog gebuik je zo:

```
namespace MyNamespace
{
  public class MyClass
  {
    //NLog recommends using a static variable for the logger object
    private static NLog.Logger logger = NLog.LogManager.GetCurrentClassLogger();

    //NLog supports several logging levels, including INFO
    logger.Info("Hello {0}", "Earth");
    try
    {
        //Do something
    }
    catch (Exception ex)
    {
        //Exceptions are typically logged at the ERROR level
        logger.Error(ex, "Something bad happened");
    }
  }
}
```

Log4NET

Log4NET is een bekende en krachtige framework waarvan er ook Log4J bestaat voor Java. Setup en configuratie is vergelijkbaar aan die van NLog. In een configuratie file worden allerlei targets gesetteld.

Log4NET gebruikt weliswaar iets genaamd *appenders.* Met deze apopenders kun je configureren om logdata op te sturen naar meerdere targets. Je kan zo kiezen hoeveel en gebaseerd op logging level wat er wordt opgestuurd naar de targets. De levels die Log4NET aanbiedt zijn hetzelfde als NLog, behalve een TRACE level.

```
private static readonly ILog log = LogManager.GetLogger(typeof(MyApp));
log.Info("Starting application.");
log.Debug("DoTheThing method returned X");
```

https://raygun.com/blog/c-sharp-logging-best-practices/

Conclusie

Ik ben erachter gekomen dat er veel op te lossen is in een korte tijd door te loggen. Aangezien ik gebruik maar van externe services kan ik mbv logging snel achterhalen wat er fout is als ik de externe service gebruik. Zo kan ik dus uit ervaring al meegeven dat ik bij mijn CoronaAPI gebruik maak van een service die het land Gambia wel in de database heeft staan maar geen data daarin mee levert. Hier kwam ik snel achter door met NLog in een try-catch te loggen. Ook kan ik hierin bekijken hoe mijn communicatie met eventuele persistentie verloopt.