**Logging in Softwareprojekten**

**1. Was ist Logging?**

* **Definition**: Logging bedeutet, dass eine Anwendung während ihrer Laufzeit Informationen über ihr Verhalten, Fehler, Zustände und Ereignisse aufzeichnet.
* **Arten von Logs**:
  + **Info-Logs** → Allgemeine Abläufe ("Benutzer X hat sich angemeldet").
  + **Debug-Logs** → Detaillierte Entwickler-Infos zur Fehlersuche.
  + **Warn-Logs** → Unerwartete Situationen, die nicht sofort kritisch sind.
  + **Error-Logs** → Fehler, die behandelt werden müssen.
  + **Fatal/ Critical** → Schwere Fehler, die das System zum Absturz bringen.

**2. Warum Logging?**

* **Fehlerdiagnose**: Probleme nachvollziehen, auch wenn sie nicht reproduzierbar sind.
* **Monitoring**: Überblick über Systemzustand und Performance.
* **Sicherheitsaspekte**: Nachvollziehbarkeit bei Angriffen oder Missbrauch.
* **Auditing**: Wer hat wann was getan? → rechtliche und betriebliche Anforderungen.
* **Transparenz im Betrieb**: Admins und DevOps sehen, was „unter der Haube“ passiert.

**3. Wie loggt man sinnvoll?**

* **Log-Ebenen gezielt einsetzen**
  + Debug → nur in der Entwicklung.
  + Info → normale Abläufe.
  + Warning → ungewöhnlich, aber tolerierbar.
  + Error → Ausnahme, muss untersucht werden.
  + Critical → System nicht mehr benutzbar.
* **Hinweise**:
  + Keine sensiblen Daten loggen (z. B. Passwörter, Kreditkarten).
  + Eindeutige Nachrichten formulieren („Fehler beim Speichern von Bestellung #1234“ statt „Fehler“).
  + Strukturierte Logs nutzen (nicht nur Text, sondern auch Felder/Werte).
  + Logs **zentralisieren**.

**4. Logging in .NET Core**

* Standardmäßig gibt es das **Microsoft.Extensions.Logging**-Framework.
* Problem: Es loggt in eine Abstraktion, aber die Ausgabe (z. B. Datei, Datenbank) muss man extra implementieren.

**5. Logging mit Serilog**

Serilog ist ein sehr beliebtes Logging-Framework für .NET.

**Vorteile von Serilog:**

* **Strukturiertes Logging**: Nicht nur Text, sondern Datenobjekte loggen.
* **Einfache Konfiguration** mit Sinks (Console, File, SQL, Elasticsearch, …).
* **Integration** in ASP.NET Core über builder.Host.UseSerilog().

**Beispiel:**

Log.Logger = new LoggerConfiguration()

.MinimumLevel.Debug()

.Enrich.FromLogContext()

.WriteTo.Console()

.WriteTo.File("logs/log.txt", rollingInterval: RollingInterval.Day)

.CreateLogger();

**Controller-Beispiel:**

public IActionResult Bestellen(int produktId)

{

\_logger.LogInformation("Bestellung gestartet für Produkt {ProduktId}", produktId);

try

{

// Logik...

return Ok("Bestellung erfolgreich!");

}

catch (Exception ex)

{

\_logger.LogError(ex, "Fehler bei Bestellung für Produkt {ProduktId}", produktId);

return StatusCode(500, "Interner Fehler!");

}

}

**Serilog-Sinks (Ausgabeziele):**

* Console → für Entwickler praktisch.
* File → einfache, persistente Logs.
* Datenbank (SQL, PostgreSQL, MongoDB …) → für zentrale Analyse.
* Cloud-Dienste (z. B. Application Insights, Seq, Elasticsearch).

**6. Fazit**

* **Was**: Logging = strukturierte Protokollierung von Ereignissen.
* **Warum**: Diagnose, Monitoring, Sicherheit, Nachvollziehbarkeit.
* **Wie**: Klare Log-Level, keine sensiblen Daten, strukturierte Logs.
* **Serilog**: Leistungsfähiges Framework für modernes, strukturiertes Logging in .NET.