htlperg
innovativ | kreativ | praxisnah

Loglevel-Synchroniser

DIPLOMARBEIT

Höhere Abteilung für Informatik

01.07.2024 – 03.04.2025

Projektmitglieder: Jan-Niclas Hartl
Maximilain Koch
Eric Reps
Luca Strobl

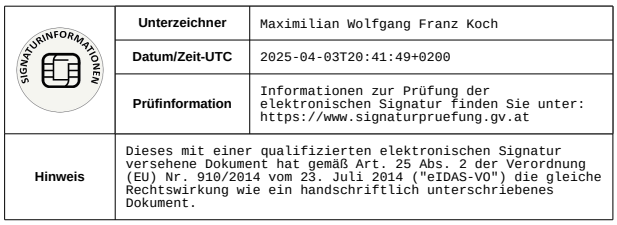
Betreuer: Dipl.-Ing.(FH) Johannes Oppitz, MSc

In Zusammarbeit mit: Programmierfabrik GmbH

Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Bei der Erstellung der Arbeit haben wir das generative KI-Tool ChatGPT zur grammatikalischen Korrektur und Rechtschreibkontrolle verwendet.



Danksagung

Wir möchten uns an dieser Stelle bei allen Menschen bedanken, die uns bei der Erstellung dieser Diplomarbeit unterstützt haben.

Dabei möchten wir uns besonders bei unserem Diplomarbeitsbetreuer Herrn Professor Dipl.-Ing.(FH) Johannes Oppitz, MSc bedanken, welcher uns bei dieser Arbeit unterstützt hat.

Ein besonderer Dank gilt auch der Programmierfabrik GmbH, die uns diese Diplomarbeit ermöglicht hat. Durch die Bereitstellung von Ressourcen, praktischen Einblicken und der Kooperation mit ihrem Team haben sie maßgeblich zu der erfolgreichen Durchführung dieses Projektes beigetragen. Ihre Unterstützung und das Vertrauen in unsere Arbeit waren eine große Hilfe. Ein besonderer Dank gilt hierbei Marko Vracar, der uns während des gesamten Prozesses betreut und uns mit wertvollen Ratschlägen und Feedback unterstützt hat.

Abstract

The company Programmierfabrik GmbH operates two servers, which share the load for various hosted sites. Each site is assigned its own logging library, defining the log levels that can be set for that site. Currently, the administrator has to manually change the log level on both servers, which is time-consuming and prone to errors. Additionally, there is a risk that a log level such as error remains active for too long, leading to excessive logging and unnecessary storage consumption.

To address these issues, this paper presents a software solution that enables centralized and simultaneous log-level updates on both servers. By automating log-level synchronization, the software reduces administrative effort and minimizes potential errors.

The solution is implemented as a web application, allowing administrators to modify log levels via a frontend. In the backend, the desired log level is stored in a Redis cache and retrieved by an external service, which then updates both servers accordingly. This ensures a consistent log-level configuration and optimizes resource usage.

Zusammenfassung

Die Programmierfabrik GmbH betreibt zwei Server, die sich die Last für verschiedene gehostete Sites teilen. Jeder dieser Sites ist eine eigene Logging-Bibliothek zugeordnet, in der definiert ist, welche Log-Level konfigurierbar sind. Derzeit muss der Administrator eine Änderung des Log-Levels manuell auf beiden Servern durchführen, was zeitaufwendig und fehleranfällig ist. Zudem besteht das Risiko, dass ein Log-Level wie „error“ zu lange aktiv bleibt, was durch übermäßiges Logging zu unnötigem Speicherverbrauch führen kann.

Um diese Probleme zu lösen, wird in dieser Arbeit eine Softwarelösung vorgestellt, die eine zentrale und gleichzeitige Aktualisierung der Log-Level auf beiden Servern ermöglicht. Durch die Automatisierung der Log-Level-Synchronisation reduziert die Software den administrativen Aufwand und minimiert potenzielle Fehler.

Die Lösung wird als Webanwendung implementiert, über die Administratoren Änderungen am Log-Level über ein Frontend vornehmen können. Im Backend wird das gewünschte Log-Level in einem Redis-Cache gespeichert und von einem externen Dienst ausgelesen. Dieser übernimmt daraufhin die Aktualisierung beider Server. Dies gewährleistet eine konsistente Log-Level-Konfiguration und optimiert die Ressourcennutzung.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

In verteilten Systemen ist eine effiziente und konsistente Log-Verwaltung essenziell, um Fehler schnell zu identifizieren und Ressourcen optimal zu nutzen. Der aktuelle manuelle Prozess zur Änderung des Log-Levels in der Programmierfabrik ist jedoch umständlich, fehleranfällig und zeitaufwendig. Administratoren müssen jede Änderung auf beiden Servern separat durchführen, was nicht nur doppelte Arbeit bedeutet, sondern auch das Risiko von Inkonsistenzen birgt.

Besonders problematisch ist, dass Log-Level häufig versehentlich über einen längeren Zeitraum auf einer ungeeigneten Stufe bleiben. Dies kann entweder dazu führen, dass wichtige Informationen verloren gehen oder unnötige Datenmengen gespeichert werden, was Speicherplatz verschwendet und die Systemperformance beeinträchtigt.

Diese Diplomarbeit setzt sich zum Ziel, eine zentrale Lösung für dieses Problem zu entwickeln. Durch eine automatisierte und synchronisierte Steuerung des Log-Levels sollen Fehler reduziert und der Administrationsaufwand gesenkt werden.

1.2 Zielsetzung

Das Ziel dieser Diplomarbeit ist es, den Prozess zur Änderung des Log-Levels in der Programmierfabrik GmbH zu optimieren. Derzeit betreibt das Unternehmen zwei Server, die sich die Last für verschiedene gehostete Sites teilen. Jeder der gehosteten Sites wird eine eigene Logging-Bibliothek zugewiesen, die spezifiziert, welche Log-Level auf der Site gesetzt werden können.

Mit der in dieser Arbeit entwickelten Software soll es möglich werden, das Log-Level zentral und gleichzeitig auf beiden Servern zu aktualisieren. Dadurch wird der Administrationsaufwand reduziert, die Fehleranfälligkeit minimiert.

1.3 Projektumfeld

1.3.1 Projektteam

Das Projektteam besteht aus vier Mitgliedern: Jan-Niclas Hartl, Maximilian Koch, Eric Reps und Luca Strobl, alle Schüler der HTL-Perg. Das Team arbeitete gemeinsam am Loglevel-Synchroniser unter der Leitung der Programmierfabrik GmbH.

1.3.2 Auftraggeber

Abbildung 1: Logo - Programmierfabrik GmbH

Der Auftraggeber dieses Projekts ist die Programmierfabrik[1], ein Unternehmen, das sich auf die Entwicklung maßgeschneiderter Softwarelösungen spezialisiert hat. Ihr Leistungsspektrum umfasst die Entwicklung von Standard- und Individualsoftware sowie umfassende Support-Dienstleistungen. Die Programmierfabrik unterstützt Unternehmen verschiedener Branchen bei der Umsetzung von IT-Projekten und bietet dabei Expertise in den Bereichen Softwareentwicklung, Datenbanken und IT-Consulting.

Im Rahmen dieses Projekts beauftragte die Programmierfabrik die Entwicklung eines Loglevel-Synchronisers. Ziel war es, eine Lösung zu schaffen, die den organisatorischen Aufwand für Administratoren reduziert und dadurch Fehlerquellen minimiert.

Während des Projekts übernahm die Programmierfabrik nicht nur die Rolle des Auftraggebers, sondern fungierte gleichzeitig als Kunde. Unser Betreuer, Marco Vracar, legte besonderen Wert darauf, uns die Realität eines Entwicklungsprozesses näherzubringen. Daher agierte er als Kunde, indem er regelmäßig Feedback gab und wertvolle Einblicke in die praktische Anwendung der Lösung gewährte. Die Zusammenarbeit zeichnete sich durch einen engen Austausch sowie eine kontinuierliche Weiterentwicklung der Zielsetzung aus.

1.3.3 Betreuung

Die Betreuung der Diplomarbeit übernahmen Herr Professor Dipl.-Ing. (FH) Johannes Oppitz, MSc., und Marco Vracar. Sie unterstützten uns während des gesamten Projekts und trugen maßgeblich dazu bei, die gesetzten Ziele erfolgreich zu erreichen.

[1] https://programmierfabrik.at

2 Theoretische Grundlagen und Technologische Konzepte

In diesem Kapitel werden die grundlegenden theoretischen Konzepte und technologischen Ansätze behandelt, die als Basis für die Entwicklung der Anwendung dienen. Die wesentlichen Grundlagen, die erforderlich sind, um die Funktionsweise der Anwendung sowie die im Frontend und Backend eingesetzten Technologien zu verstehen, werden erläutert.

2.1 Theoretische Grundlagen

2.1.1 Integrated Development Environment (IDE)

Eine IDE[2] ist eine Software, die zentrale Entwicklungsfunktionen wie Codebearbeitung, Kompilierung, Testen und Debugging in einer Anwendung vereint. Sie steigert die Produktivität, indem sie automatisierte Werkzeuge zur Verfügung stellt, die den Entwicklungsprozess effizienter und fehlerärmer gestalten.

2.1.2 Logging

Logging umfasst das systematische Erfassen und Speichern von Ereignissen, die während des Betriebs einer Anwendung auftreten. Diese Ereignisse werden in Logeinträgen festgehalten und können detaillierte Informationen über den Anwendungsablauf, aufgetretene Fehler sowie Warnungen und andere relevante Betriebszustände enthalten. Zentral dabei sind die sogenannten LogLevel, die den Schweregrad eines Ereignisses klassifizieren. Beispielsweise werden detaillierte Debug-Informationen, die vorwiegend zur Fehlersuche während der Entwicklung benötigt werden, unter dem LogLevel Debug abgelegt, wohingegen kritische Fehler unter den LogLeveln Error oder Fatal geführt werden. Durch diese Klassifikation ist es möglich, die Logdaten zu filtern und den Fokus auf die für den jeweiligen Analysezweck wichtigen Informationen zu richten. Die strukturierte Protokollierung, bei der Logeinträge in Form von strukturierten Formaten wie JSON abgelegt werden, erlaubt es zudem, die Daten maschinenlesbar und somit effizient

[2] https://aws.amazon.com/what-is/ide

mit automatisierten Tools auszuwerten. Darüber hinaus liefern Logdaten wichtige Hinweise für sicherheitsrelevante Untersuchungen, indem sie den Verlauf von Benutzeraktionen und Systemzugriffen nachvollziehbar dokumentieren (vgl. [1]).

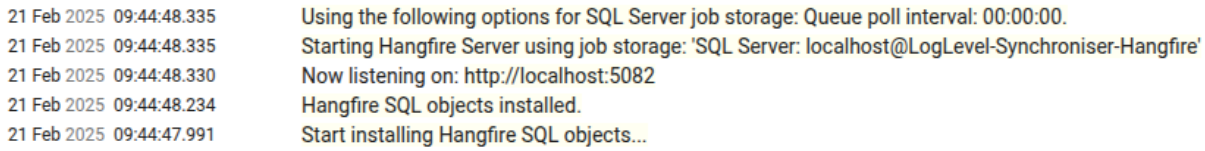


Abbildung 2: Beispiel - Logs in Seq

Die Abbildung 2 veranschaulicht exemplarisch mehrere Logeinträge, die in Seq, welches im Abschnitt 2.2.1 näher erläutert wird, visualisiert werden. Es wird dabei nicht im Detail auf die einzelnen Inhalte der Logeinträge eingegangen; vielmehr soll demonstriert werden, dass Logdaten generiert und in Seq erfasst werden. Diese Abbildung belegt somit die erfolgreiche Umsetzung des Logging-Konzepts, das den Betrieb der Anwendung dokumentiert und als Grundlage für weitere Überwachungs- und Analyseprozesse dient. Im folgenden Abschnitt 2.2.1 werden ergänzend vertiefte Einblicke in Seq sowie dessen spezifische Funktionsweise vermittelt.

2.1.3 OpenTelemetry Protocol (OTLP)

OTLP, das OpenTelemetry-Protokoll[3], dient als standardisierte Methode zur Übertragung von Telemetriedaten – wie Leistungs- und Fehlerinformationen – an zentrale Überwachungssysteme wie Seq (siehe Abschnitt 2.2.1). Durch diese einheitliche Formatierung wird sichergestellt, dass die Daten konsistent und effizient analysiert werden können. Mit OTLP können verschiedene Anwendungen ihre Betriebsdaten in einem gemeinsamen Format übermitteln, was die Überwachung der Systemleistung vereinfacht und eine schnelle Identifikation von Problemen ermöglicht.

2.1.4 Object-Relational Mapping (ORM)

ORM ist ein Konzept, das die Verbindung zwischen objektorientierter Programmierung und relationalen Datenbanken erleichtert, indem es eine Abstraktionsschicht über die Datenbankinteraktionen legt. Anstatt direkte SQL-Abfragen zu formulieren, können Entwickler mit Daten in Form von Objekten arbeiten, die den Strukturen der Datenbank entsprechen. Ein ORM-Framework übernimmt dabei die Übersetzung zwischen den objektorientierten Datenmodellen der Anwendung und den relationalen Tabellen der Datenbank. Dabei entspricht jede

[3] https://opentelemetry.io/docs/what-is-opentelemetry

Klasse einer Tabelle, jede Instanz dieser Klasse einer Zeile und jedes Attribut einer Spalte der Datenbank.

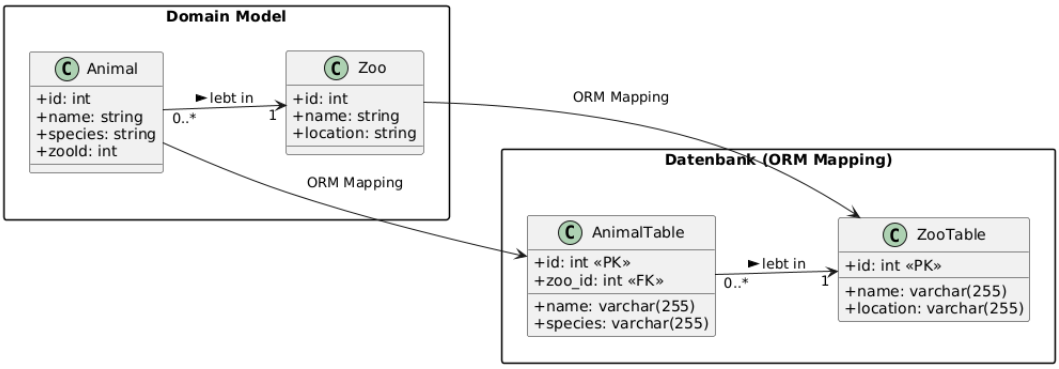


Abbildung 3: ORM-Beispiel - Animal & Zoo

Wie in Abbildung 3 dargestellt, wird hier gezeigt, wie das objektorientierte Modell in relationale Tabellen übersetzt wird. Die Klassen Animal und Zoo entsprechen den Tabellen AnimalTable und ZooTable. Dabei enthält die Klasse Animal das Attribut zooId, das als Fremdschlüssel (zoo_id) auf die ZooTable verweist und die Beziehung „Ein Tier lebt in genau einem Zoo“ abbildet.

2.1.5 Migrations

Migrations bestehen aus einer Reihe von Dateien, die schrittweise Anpassungen an der DB-Struktur dokumentieren, z. B. das Hinzufügen oder Entfernen von Tabellen und Spalten. Jede Migration umfasst dabei in der Regel zwei Komponenten: eine „Up-Migration“, die die gewünschten Änderungen implementiert, und eine „Down-Migration“, die diese rückgängig macht. Dadurch bleibt die DB-Struktur versioniert, was insbesondere bei der Zusammenarbeit in Teams von Vorteil ist. Migrations bieten somit eine automatisierte, kontrollierte und sichere Möglichkeit, die DB konsistent mit dem Code weiterzuentwickeln und Fehler zu minimieren, da alle Änderungen strukturiert und schrittweise durchgeführt werden.

```
1 Add-Migration AddAnimal
```

Listing 1: Erstellen der Migration AddAnimal

Listing 1 zeigt den PowerShell-Befehl Add-Migration AddAnimal. Mit diesem Befehl wird eine neue Migration erstellt, die die anstehenden Änderungen an der Datenbank dokumentiert. In diesem Fall wird eine Migration erzeugt, die die Tabelle Animals anlegt. Zur besseren Übersichtlichkeit wurden die Spaltendefinitionen in der generierten Migration weggelassen.

```
public partial class AddAnimal : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(name: "Animals");
    }

    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(name: "Animals");
    }
}
```

Listing 2: C# Migration Beispiel: Hinzufügen der Tabelle Animals

Listing 2 zeigt den C#-Code der Migration, die durch den Befehl in Listing 1 erstellt wurde. Die Methode Up implementiert das Anlegen der Tabelle Animals, während die Methode Down diese Änderung rückgängig macht. Diese Struktur ermöglicht es, Änderungen an der Datenbank einfach vorwärts oder rückwärts zu migrieren.

```
Update-Database
```

Listing 3: Ausführen der Migration (Update-Database)

Listing 3 zeigt den PowerShell-Befehl Update-Database. Mit diesem Befehl wird die aktuellste, noch nicht ausgeführte Migration angewendet – in diesem Fall wird die Up-Methode der Migration AddAnimal ausgeführt, wodurch die Tabelle Animals in der Datenbank angelegt wird.

```
Update-Database Initial
```

Listing 4: Zurücksetzen auf die ursprüngliche Datenbankversion (Initial)

Listing 4 zeigt den PowerShell-Befehl Update-Database Initial. Dieser Befehl wird verwendet, um die Datenbank auf eine zuvor definierte Ausgangsversion zurückzusetzen. Dabei wird die Down-Methode der Migration ausgeführt, wodurch alle in der Migration AddAnimal vorgenommenen Änderungen rückgängig gemacht werden.

2.1.6 Representational State Transfer (REST)

REST ist ein Architekturstil für Webservices, der eine einfache, skalierbare und lose gekoppelte Kommunikation ermöglicht. Es basiert auf dem Client-Server-Modell und nutzt das HTTP-Protokoll mit standardisierten Methoden wie GET, POST, PUT und DELETE.

Ein wesentliches Merkmal ist die Statelessness, d. h., jede Anfrage enthält alle nötigen Informationen, ohne dass der Server Sitzungsdaten speichert. Dadurch wird die Skalierbarkeit erhöht. REST nutzt Ressourcen, die über URLs identifiziert werden, und gibt Antworten oft im JSON- oder XML-Format zurück.

```
curl "https://api.ipify.org?format=json"
```

Listing 5: REST Beispiel: Abrufen der öffentlichen IP-Adresse mit cURL

Listing 5 zeigt ein Beispiel für den Einsatz von cURL, einem Kommandozeilen-Werkzeug, das einen GET-Request an einen Web-Service sendet. In diesem Fall wird eine Anfrage an die API von ipify geschickt, die die öffentliche IP-Adresse des Rechners im JSON-Format zurückliefert:

```
{"ip":"123.45.67.89"}
```

Dieses Beispiel verdeutlicht, wie REST-Anfragen aufgebaut sind und wie standardisierte HTTP-Methoden und -Formate die Interaktion zwischen Client und Server vereinfachen.

2.1.7 Mocking

In der Softwareentwicklung versteht man unter Mocking[4] das Erstellen von Scheinobjekten, die das Verhalten realer Objekte nachahmen. Diese sogenannten Mock-Objekte werden hauptsächlich in Unit-Tests eingesetzt, um einzelne Komponenten isoliert von ihren Abhängigkeiten zu testen. Dabei simulieren sie die Interaktionen mit den tatsächlichen Objekten, ohne deren vollständige Implementierung zu benötigen. Ein Beispiel für den Einsatz von Mocking in der Anwendung ist in Listing 64 zu sehen, wo in einem Unit-Test das Verhalten eines Service-Mock-Objekts simuliert wird.

2.1.8 Data transfer object (DTO)

Ein Data Transfer Object[5] (DTO) ist ein einfaches Objekt, das primär dazu dient, Daten zwischen verschiedenen Schichten oder Systemen auszutauschen. DTOs enthalten keinerlei Geschäftslogik, sondern lediglich Felder, in denen Daten gespeichert werden. Durch diese Entkopplung wird verhindert, dass die interne Datenstruktur einer Anwendung direkt an externe Schnittstellen gebunden wird. Insbesondere in verteilten Systemen, in denen Netzwerkaufrufe teuer sein können, ermöglichen DTOs das Bündeln mehrerer Werte in einem einzigen Objekt, wodurch die Anzahl der erforderlichen Aufrufe reduziert und die Performance verbessert wird. Ein praktisches

[4] https://stackoverflow.com/a/2666006
[5] https://martinfowler.com/eaaCatalog/dataTransferObject.html

Beispiel, das die Anwendung eines DTOs in dieser Arbeit veranschaulicht, wird in Listing 15 gezeigt. Dort wird das DTO UpdateSiteLogLevelDto verwendet, um Daten zwischen den Schichten auszutauschen.

2.2 Backend-Technologien & Bibliotheken

2.2.1 Seq

Abbildung 4: Logo – Seq [6]

Abbildung 5 zeigt die Web-Oberfläche von Seq, einem zentralen Log-Management-System. Anhand dieser Ansicht wird deutlich, wie Seq strukturierte Logdaten sammelt, speichert und visualisiert. Die Oberfläche ermöglicht eine schnelle Analyse und Filterung von Logeinträgen, was insbesondere bei der Fehlerdiagnose und Überwachung des Anwendungsbetriebs von großem Vorteil ist.

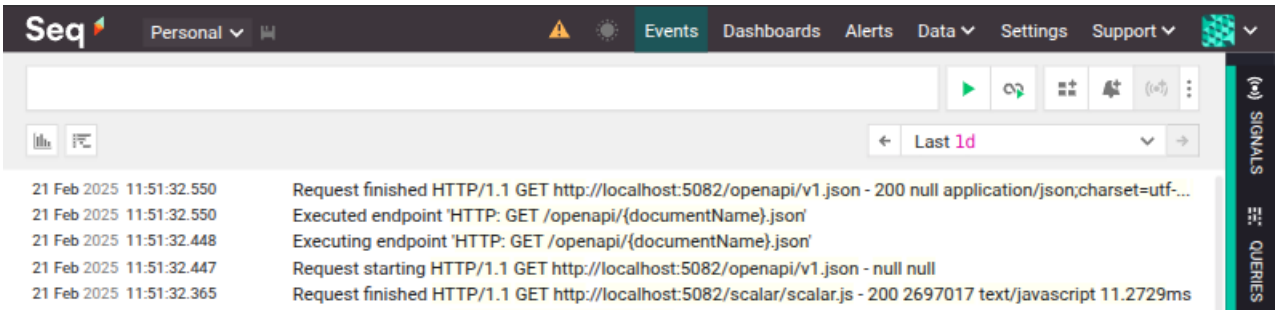


Abbildung 5: Seq - Web-Oberfläche

2.2.2 Entity Framework Core (EF Core)

Abbildung 6: Logo – Entity Framework [7]

[6] https://datalust.co/img/seq-logo-dark.svg

Das Entity Framework Core[8] wurde als ORM-Framework eingesetzt, um die Interaktion zwischen der Anwendung und der relationalen Datenbank zu verwalten. EF Core ist ein leichtgewichtiges, plattformübergreifendes und leistungsstarkes ORM-Framework von Microsoft, das die Durchführung von Datenbankoperationen in einer .NET-Umgebung vereinfacht und optimiert. Die in Abschnitt 2.1.4 erläuterten Grundlagen des ORM-Konzepts bilden die theoretische Basis für diese Vorgehensweise.

2.2.3 ASP.NET Core Web API

Abbildung 7: Logo – .NET Core [9]

ASP.NET Core[10] wird in dieser Arbeit für die Implementierung des Backends der Webanwendung eingesetzt. Das plattformübergreifende Open-Source-Webframework von Microsoft ist speziell für die Entwicklung moderner Webanwendungen und APIs optimiert. Da die Implementierung als Web API erfolgt, kommt das REST-Prinzip zur Anwendung, dessen Grundlagen in Abschnitt 2.1.6 erläutert sind.

2.2.4 Remote Dictionary Server (Redis)

Abbildung 8: Logo - Redis [11]

Redis ist ein In-Memory-Speicher, der in vielen Anwendungsfällen wie Caching, Sitzungsdaten, Echtzeit-Datenanalysen und Messaging eingesetzt wird. Er entstand 2009 durch die Entwicklung

[7] https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore/9.0.2/icon
[8] https://learn.microsoft.com/en-us/ef/core
[9] https://commons.wikimedia.org/wiki/File:.NET_Core_Logo.svg
[10] https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core
[11] https://www.svgrepo.com/show/303460/redis-logo.svg

von Salvatore Sanfilippo. Im Gegensatz zu klassischen, vorwiegend festplattenbasierten relationalen Datenbanksystemen (z. B. MySQL, PostgreSQL, Oracle) arbeitet Redis überwiegend im RAM (Random Access Memory). Dadurch können Anfragen schnell beantwortet werden, da die Zugriffszeiten im Hauptspeicher deutlich kürzer sind als auf Festplatten oder SSDs. Zusätzlich stellt Redis Persistenzmechanismen zur Verfügung, um Daten dauerhaft zu speichern (vgl. [2]).

2.2.5 MSSQL

Abbildung 9: Logo - MSSQL [12]

Microsoft SQL Server[13] ist ein relationales Datenbankmanagementsystem (RDBMS), das Daten speichert, verwaltet und verarbeitet. Es verwendet Transact-SQL (T-SQL) für Abfragen und unterstützt verschiedene Bereitstellungsoptionen, darunter lokale Installationen, Container und Cloud-Dienste wie Azure SQL Database. Die Datenbank-Engine bildet das Kernstück und bietet hohe Skalierbarkeit sowie Sicherheitsfunktionen wie Verschlüsselung und rollenbasierte Zugriffskontrolle.

2.2.6 Serilog

Abbildung 10: Logo - Serilog [14]

Serilog[15] ist eine weit verbreitete Logging-Bibliothek für .NET-Anwendungen. Im Gegensatz zu herkömmlichen Logging-Frameworks, die oft nur unstrukturierte Textnachrichten erzeugen,

[12] https://www.svgrepo.com/show/303229/microsoft-sql-server-logo.svg
[13] https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server
[14] https://api.nuget.org/v3-flatcontainer/serilog/4.2.1-dev-02340/icon
[15] https://serilog.net/

ermöglicht Serilog das sogenannte strukturierte Logging. Das bedeutet, dass Logeinträge als Datensätze mit einzelnen Attributen (wie Zeitstempel, Log-Level, Ereignisbeschreibung und weiteren Kontextinformationen) gespeichert werden. Dies erleichtert später das Filtern, Suchen und Analysieren der Log-Daten erheblich. Außerdem bietet Serilog eine Vielzahl von sogenannten Sinks, also Zielsystemen, in die die Logeinträge weitergeleitet werden können – etwa in Konsolen, Dateien oder spezialisierte Log-Management-Systeme. Die Konfiguration von Serilog in der Anwendung ist in Listing 42 dargestellt. Dabei erfolgt die Übertragung der Logeinträge unter Verwendung des OTLP-Protokolls an den entsprechenden OTLP-Endpunkt. Eine Erläuterung des OTLP-Protokolls ist in Abschnitt 2.1.3 zu finden.

2.2.7 FluentResults

Abbildung 11: Logo - FluentResults [16]

FluentResults[17] ist eine Bibliothek, die das Management von Erfolgs- und Fehlerzuständen in .NET-Anwendungen vereinfacht. Anstatt Exceptions in jedem Fall zu verwenden, erlaubt FluentResults, das Ergebnis einer Operation explizit als Erfolg oder Misserfolg zu deklarieren. Dies führt zu einem klareren und nachvollziehbareren Code, da Fehler nicht über Ausnahmen weitergegeben werden, sondern als Bestandteil des Rückgabewerts behandelt werden. Mit einer flüssigen, sogenannten Fluent API können Entwickler zusätzliche Fehlerinformationen oder Warnungen anhängen, was die Fehlerbehandlung deutlich flexibler gestaltet.

```
1 public Result<double> Divide(double numerator, double denominator)
2 {
3     if (denominator == 0)
4         return Result.Fail<double>("Division durch Null ist nicht erlaubt.");
5     var result = numerator / denominator;
6     return Result.Ok(result);
7 }
```

Listing 6: FluentResults Beispiel - Division durch 0

Die Methode Divide in Listing 6 demonstriert den Einsatz von FluentResults zur Handhabung von Erfolgs- und Fehlerzuständen. Wird als Divisor der Wert 0 übergeben, erfolgt die Rückgabe

[16] https://api.nuget.org/v3-flatcontainer/fluentresults/3.16.0/icon
[17] https://github.com/altmann/FluentResults

eines Fehlers mittels Result.Fail. Andernfalls wird das Ergebnis der Division mittels Result.Ok als Erfolg deklariert.

2.2.8 xUnit

Abbildung 12: Logo - xUnit [18]

xUnit[19] ist ein kostenloses und offenes Unit-Test-Framework für das .NET-Framework. Es wurde von den ursprünglichen Entwicklern von NUnit v2 geschrieben und stellt die neueste Technologie für das Unit-Testing von C#, F#, VB.NET und anderen .NET-Sprachen dar. xUnit.net ist mit verschiedenen Entwicklungsumgebungen kompatibel, darunter Visual Studio, Visual Studio Code, ReSharper/Rider, CodeRush und NCrunch. Es ist Teil der .NET Foundation und unter der Apache 2-Lizenz veröffentlicht.

2.2.9 Moq

Abbildung 13: Logo - Moq [20]

Moq[21] ist ein populäres Mocking-Framework für .NET, das entwickelt wurde, um die Erstellung von Mock-Objekten zu erleichtern. Es nutzt LINQ-Ausdrucksbäume und Lambda-Ausdrücke, um eine produktive, typsichere und refaktorisierungsfreundliche API bereitzustellen. Mit Moq können sowohl Schnittstellen als auch Klassen gemockt werden. Die Grundlagen des Mockings sind in Abschnitt 2.1.7 erläutert.

[18] https://api.nuget.org/v3-flatcontainer/xunit/2.9.3/icon
[19] https://xunit.net
[20] https://api.nuget.org/v3-flatcontainer/moq/4.20.72/icon
[21] https://github.com/devlooped/moq

2.2.10 StackExchange.Redis

StackExchange.Redis[22] ist eine leistungsstarke, von Stack Overflow entwickelte .NET-Bibliothek zur Interaktion mit Redis. Sie zeichnet sich durch ihre hohe Effizienz, Multiplexing-Fähigkeit (gleichzeitige Nutzung einer Verbindung durch mehrere Threads) und Unterstützung für synchrones/asynchrones Programmieren aus. Im Folgenden wird exemplarisch gezeigt, wie StackExchange.Redis in der Anwendung integriert wurde (siehe Listing 23).

2.2.11 Newtonsoft.Json

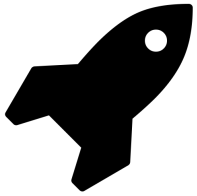


Abbildung 14: Logo - Newtonsoft.Json [23]

Newtonsoft.Json[24], auch bekannt als Json.NET, ist eine weit verbreitete JSON-Bibliothek für .NET-Anwendungen. Sie ermöglicht das einfache Serialisieren von .NET-Objekten in das JSON-Format sowie das Deserialisieren von JSON-Daten zurück in .NET-Objekte. Diese Funktionalität ist besonders nützlich, um Daten zwischen verschiedenen Systemen auszutauschen oder Konfigurationsdaten in einer leicht lesbaren und strukturierten Form zu speichern. Newtonsoft.Json ist aufgrund seiner einfachen API, hohen Leistungsfähigkeit und umfangreichen Konfigurationsmöglichkeiten zu einem Standard in vielen .NET-Projekten geworden. In der Anwendung wird Newtonsoft.Json zur Serialisierung von DTOs verwendet. In Listing 23 wird beispielsweise in Zeile 6 das DTO, das in Listing 19 dargestellt wird, in das JSON-Format umgewandelt.

2.2.12 Hangfire

Abbildung 15: Logo - Hangfire [25]

[22] https://stackexchange.github.io/StackExchange.Redis
[23] https://api.nuget.org/v3-flatcontainer/newtonsoft.json/13.0.3/icon
[24] https://www.newtonsoft.com/json
[25] https://api.nuget.org/v3-flatcontainer/hangfire/1.8.18/icon

Hangfire[26] ist eine Open-Source-Bibliothek für .NET-Anwendungen, die die Verarbeitung von Hintergrundaufgaben erheblich vereinfacht. Mit Hangfire können Aufgaben, die nicht unmittelbar im Hauptanwendungsfluss ausgeführt werden müssen, asynchron im Hintergrund abgearbeitet werden. Dies schließt verschiedene Arten von Jobs ein, wie etwa Fire-and-Forget-Aufgaben, bei denen eine Aufgabe einmalig gestartet wird, verzögerte Jobs, die zu einem späteren Zeitpunkt ausgeführt werden, wiederkehrende Aufgaben, die in regelmäßigen Abständen laufen, sowie Aufgabenkontinuitäten, bei denen eine Aufgabe erst gestartet wird, wenn eine vorherige erfolgreich abgeschlossen wurde. Ein praktischer Anwendungsfall von Hangfire ist in Listing 28 zu sehen.

2.3 Dokumentation und API-Entwicklung

2.3.1 Swagger

Abbildung 16: Logo - Swagger [27]

Swagger[28] ist ein Open-Source-Projekt, das die Beschreibung und Dokumentation von RESTful APIs ermöglicht. Durch die Definition einer API-Struktur in maschinenlesbarer Form können zahlreiche Vorteile realisiert werden. So ermöglicht Swagger die automatische Generierung von interaktiver API-Dokumentation, die Erstellung von Client-Bibliotheken in verschiedenen Programmiersprachen sowie die Unterstützung bei automatisierten Tests. Die API-Spezifikation wird dabei in einer YAML- oder JSON-Datei festgehalten, die detaillierte Informationen über die verfügbaren Operationen, Parameter, Rückgabewerte und Authentifizierungsmechanismen enthält. Diese Spezifikation kann entweder manuell erstellt oder automatisch aus Anmerkungen im Quellcode generiert werden.

[26] https://www.hangfire.io
[27] https://www.svgrepo.com/show/354420/swagger.svg
[28] https://swagger.io/docs/specification/v2_0/what-is-swagger

Abbildung 17: Endpunkte in Swagger

Abbildung 17 zeigt ein Beispiel für eine mit Swagger generierte API-Dokumentation. Hier werden die verfügbaren Endpunkte (z.B. /api/sites und /api/sites/{siteId}/LogLevels) übersichtlich dargestellt. Zudem lassen sich Parameter sowie Rückgabewerte direkt im Browser testen, was die Entwicklung und Wartung einer RESTful API deutlich vereinfacht.

2.4 Frontend-Technologien & Bibliotheken

2.4.1 Python

Abbildung 18: Logo - Python [29]

Python[30] ist eine interpretierte, objektorientierte und dynamisch typisierte Programmiersprache mit einfacher Syntax und hoher Lesbarkeit. Sie unterstützt Module und Pakete zur modularen Programmierung und verfügt über eine umfangreiche Standardbibliothek. Als plattformunabhängige Sprache ermöglicht sie einen Entwicklungsprozess ohne Kompilierung.

[29] https://www.svgrepo.com/show/452091/python.svg
[30] https://www.python.org/doc/essays/blurb

2.4.2 Flask

Abbildung 19: Logo - Flask [31]

Flask[32] ist ein einfaches Webframework für Python, mit dem sich Webanwendungen erstellen lassen. Es basiert auf WSGI[33], einem Standard, der regelt, wie Webserver mit Python-Anwendungen kommunizieren. Flask stellt essenzielle Funktionen für den Aufbau von Webseiten und APIs bereit und verzichtet dabei bewusst auf restriktive Vorgaben, um größtmögliche Flexibilität zu ermöglichen. Für die Gestaltung von HTML-Seiten kommt das Template-System Jinja2 zum Einsatz.

2.4.3 Jinja2

Abbildung 20: Logo - Jinja [34]

Jinja[35] ist eine schnelle und erweiterbare Template Engine für Python, die es ermöglicht, dynamische Inhalte aus Daten zu erzeugen. Eine Template Engine ist ein System, das Vorlagen mit Platzhaltern und Steueranweisungen nutzt, um strukturierte Ausgaben wie HTML, XML oder andere textbasierte Formate zu generieren. Sie trennt die Präsentationsebene von der Programmlogik, indem sie Platzhalter und Kontrollstrukturen in Templates erlaubt. Ein praktischer Anwendungsfall von Jinja ist in Listing 45 zu sehen

[31] https://www.svgrepo.com/show/508915/flask.svg
[32] https://flask.palletsprojects.com/en/stable
[33] https://builtin.com/data-science/wsgi
[34] https://www.svgrepo.com/show/473669/jinja.svg
[35] https://jinja.palletsprojects.com/en/stable/intro

2.4.4 W3.CSS

W3.CSS[36] ist ein modernes, reines CSS-Framework für responsive Webdesign. Es ist leichtgewichtig, einfach zu erlernen und bietet vordefinierte Klassen für Farben, Layouts und Navigationselemente. W3.CSS benötigt keine zusätzlichen Bibliotheken und stellt plattformübergreifende Kompatibilität sicher.

2.4.5 Pip Installs Packages

Pip[37] ist das zentrale Paketverwaltungssystem für Python und ermöglicht die Installation, Aktualisierung sowie Verwaltung von Bibliotheken und Abhängigkeiten. Es bietet direkten Zugriff auf das Python Package Index (PyPI), eine umfassende Sammlung von Open-Source-Paketen. Zudem unterstützt es die Verwaltung von Abhängigkeiten innerhalb virtueller Umgebungen und erlaubt die Konfiguration zur Nutzung benutzerdefinierter Repositories als Paketquelle.

```
1 # Installation von Flask aus dem offiziellen PyPI-Repository
2 pip install flask
3
4 # Aktualisierung eines installierten Pakets
5 pip install --upgrade flask
6
7 # Installation eines Pakets aus einer requirements-Datei
8 pip install -r requirements.txt
```

Listing 7: Pip – Paketinstallation, Upgrade und Installation aus requirements.txt

In Listing 7 wird die grundlegende Nutzung von Pip anhand des Web-Frameworks Flask demonstriert. Neben der Installation von Flask aus dem offiziellen PyPI-Repository (pip install flask) zeigt das Beispiel, wie eine Aktualisierung eines bereits installierten Pakets (pip install –upgrade flask) oder die Installation mehrerer Abhängigkeiten aus einer requirements.txt-Datei erfolgen kann. Letzteres ist besonders hilfreich für die Verwaltung und Reproduzierbarkeit von Entwicklungsumgebungen.

[36] https://www.w3schools.com/w3css
[37] https://packaging.python.org/en/latest/guides/tool-recommendations

2.4.6 jQuery

Abbildung 21: Logo - jQuery [38]

jQuery[39] ist eine weit verbreitete JavaScript-Bibliothek, die speziell dazu entwickelt wurde, die Interaktion mit HTML-Dokumenten zu vereinfachen. Sie ermöglicht es, das Document Object Model (DOM) effektiv zu durchsuchen und zu manipulieren, wodurch das Einbinden von Effekten, Animationen und Ajax-Interaktionen wesentlich einfacher wird. Durch die abstrahierte und intuitive API von jQuery können Entwickler mit relativ wenig Code komplexe Aufgaben lösen – und das browserübergreifend einheitlich.

2.4.7 Asynchronous Javascript and XML (AJAX)

AJAX[40] ist in der Webprogrammierung ein Konzept, welches den asynchronen Datenaustausch zwischen dem Webbrowser und einem Backend-Server beschreibt. Mithilfe dieses Konzeptes, kann der Browser Datenabfragen, also HTTP-Abfragen an das Backend schicken, die Daten erhalten und im Webinterface anzeigen, ohne dass der Browser die gesamte Seite neu laden muss.

[38] https://www.svgrepo.com/show/353940/jquery.svg
[39] https://jquery.com/
[40] https://de.wikipedia.org/wiki/Ajax_(Programmierung)

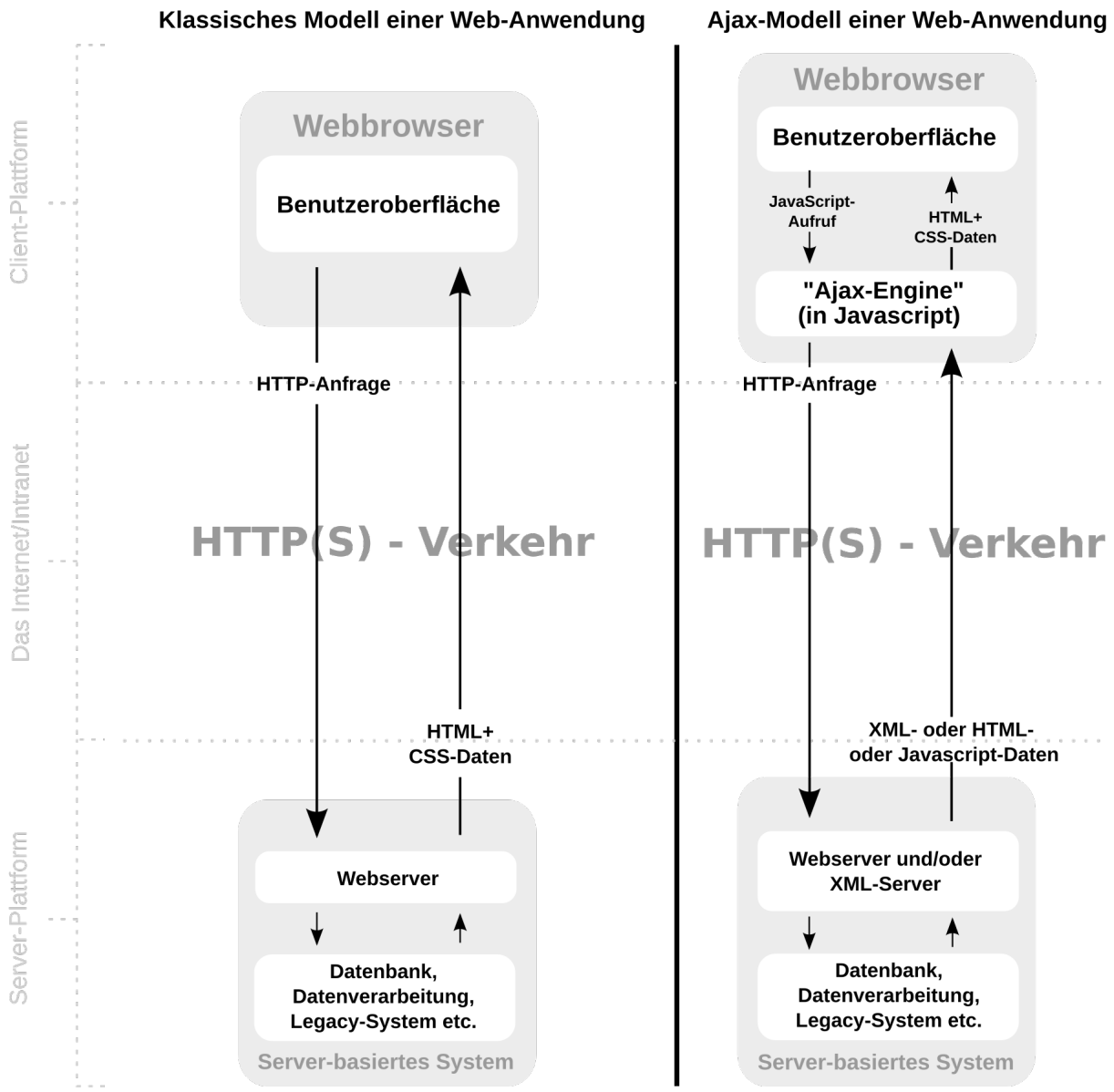


Abbildung 22: Vergleich AJAX mit klassischem Datenverkehr [41]

Wie man in Abbildung 22 sieht, verwendet der Webbrowser, wenn man AJAX verwendet, zusätzlich zur Benutzeroberfläche noch eine AJAX-Engine, welche sich um die HTTP-Abfragen kümmert, sodass die Browserseite nicht komplett neu geladen werden muss, sondern nur an spezifischen Stellen, das Document Object Model (DOM) geändert wird.

[41] https://upload.wikimedia.org/wikipedia/commons/thumb/d/d8/Ajax-vergleich.svg/1920px-Ajax-vergleich.svg.png

2.5 Entwicklungswerkzeuge und Paketmanager

2.5.1 Pycharm

Abbildung 23: Logo - Pycharm [42]

PyCharm[43] ist eine integrierte Entwicklungsumgebung (IDE) für Python, entwickelt von JetBrains, die intelligente Codevervollständigung, Echtzeit-Fehlerprüfung und schnelle Korrekturen bietet. Sie unterstützt Webentwicklung und ist plattformübergreifend für Windows, macOS und Linux verfügbar.

2.5.2 Visual Studio

Abbildung 24: Logo - Visual Studio [44]

Visual Studio[45] ist eine umfassende IDE (siehe 2.1.1), die es ermöglicht, den gesamten Entwicklungsprozess zu bearbeiten. Sie bietet Funktionen wie Compiler, Codevervollständigung, Debugging-Tools und Versionskontrolle. Mit Visual Studio können Anwendungen in verschiedenen Programmiersprachen entwickelt und getestet werden. Die hohe Flexibilität dieser Entwicklungsumgebung beruht maßgeblich auf dem breiten Spektrum an verfügbaren Erweiterungen für Visual Studio. Diese ermöglichen eine gezielte Anpassung der Entwicklungsumgebung an individuelle Anforderungen und spezifische Anwendungsfälle. Durch die Integration von Git, GitHub und Azure DevOps können Entwickler direkt in der IDE Versionskontrolle und Code-Reviews durchführen.

[42] https://www.svgrepo.com/show/354237/pycharm.svg
[43] https://www.jetbrains.com/pycharm
[44] https://www.svgrepo.com/show/354520/visual-studio.svg
[45] https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide

2.5.3 Nuget

Abbildung 25: Logo - Nuget [46]

Nuget[47] ist der Paketmanager für die .NET-Plattform und erleichtert Entwicklern das Verwalten von Abhängigkeiten in ihren Projekten. Mit NuGet können Bibliotheken und Tools effizient gefunden, installiert und aktualisiert werden. Pakete werden über ein zentrales Repository bereitgestellt, das sowohl öffentliche als auch private Pakete unterstützt.

2.6 Versionskontrolle und DevOps

2.6.1 Git

Abbildung 26: Logo - Git [48]

Git ist ein verteiltes Versionskontrollsystem, das Entwicklern ermöglicht, den gesamten Verlauf von Änderungen an Projekten nachzuvollziehen, effizient zusammenzuarbeiten und Konflikte zu vermeiden. Git speichert den Zustand von Dateien als Snapshots, was schnelle Zugriffe auf frühere Versionen ermöglicht. Es unterstützt die gleichzeitige Arbeit mehrerer Entwickler an einem Projekt, da jeder eine eigene Kopie des Repositories besitzt. So kann auch offline gearbeitet werden und Änderungen können später synchronisiert werden (vgl. [3])

Ein großer Vorteil von Git ist seine Flexibilität und Geschwindigkeit. Die meisten Operationen werden lokal ausgeführt, was das Arbeiten mit großen Codebasen beschleunigt. Zudem bietet Git umfassende Möglichkeiten zur Zusammenarbeit, zum Beispiel durch Branches. Ein Entwickler kann in einem Branch neue Funktionen entwickeln, ohne den Hauptcode zu beeinträchtigen,

[46] https://www.svgrepo.com/show/373937/nuget.svg
[47] https://www.nuget.org
[48] https://www.svgrepo.com/show/452210/git.svg

und später die Änderungen sicher in den Hauptzweig (main oder früher als master bezeichnet) integrieren.

```
# In das Projektverzeichnis wechseln
cd LogLevel-Synchroniser

# In den Feature-Branch wechseln
git checkout 20200054/controllers

# Status des Repositories überprüfen
git status

# Die bearbeiteten Dateien zum Staging hinzufügen
git add .

# Die Änderungen mit einer aussagekräftigen Nachricht committen
git commit -m "Refactored logging mechanism in SiteController.cs"

# Änderungen ins Repository hochladen
git push
```

Listing 8: Git-Workflow - Branch-Wechsel, Commit & Push

Das in Listing 8 dargestellte Kommandozeilenbeispiel zeigt einen typischen Git-Workflow für die Versionierung und Verwaltung von Änderungen in einem separaten Feature-Branch. Nach dem Wechsel in das Projektverzeichnis und den entsprechenden Branch werden Änderungen an der Datei SiteController.cs erfasst, versioniert und in das Remote-Repository übertragen.

2.6.2 Azure DevOps

Azure DevOps ist eine Sammlung von Tools, die Entwicklern, Projektmanagern und anderen Beteiligten helfen, Software schneller und effizienter zu entwickeln. Es unterstützt die Zusammenarbeit im Team und bietet Funktionen wie Versionskontrolle, Continuous Integration (CI), und Projektmanagement. Man kann Azure DevOps entweder in der Cloud (Azure DevOps Services) oder lokal auf eigenen Servern (Azure DevOps Server) nutzen. Es bietet eine integrierte Umgebung, die man über den Webbrowser oder IDEs wie Visual Studio verwenden kann, um den gesamten Entwicklungsprozess zu verwalten (vgl. [4])

3 Projektplanung

In diesem Kapitel wird die Planung des Projekts detailliert dargestellt. Es werden die Ziele, der zeitliche Ablauf, die Meilensteine sowie die eingesetzten Ressourcen beschrieben. Die Projektplanung bildet die Grundlage dafür, dass die im weiteren Verlauf der Arbeit vorgestellten Implementierungen strukturiert und zielgerichtet umgesetzt werden können.

3.1 Teamorganisation und agile Arbeitsmethoden

Daily Stand-up

Ein kurzes Daily Stand-up (ca. 5 bis 10 Minuten) diente dazu, den aktuellen Stand der Arbeiten zu besprechen, Probleme schnell zu lösen und die Prioritäten für den nächsten Arbeitstag festzulegen. Diese kurzen Meetings halfen, Entscheidungen zügig zu treffen und Informationen effizient auszutauschen.

Weekly Retrospective

Neben den täglichen Abstimmungen fand einmal pro Woche eine Reflexion statt. Dabei wurden die Erfahrungen der vergangenen Woche besprochen und Möglichkeiten zur Verbesserung der Arbeitsabläufe identifiziert. Diese regelmäßigen Feedbackrunden ermöglichten es, die Prozesse kontinuierlich anzupassen.

Direkte Kommunikation

Da wir alle im selben Büro arbeiteten, war ein persönlicher Austausch jederzeit möglich. Wenn unser Betreuer, Herr Marko Vracar, nicht anwesend war, nutzten wir Microsoft Teams, um die Kommunikation fortzusetzen.

Schrittweise Abstimmung mit dem Kunden

Zu Beginn jeder Projektphase vermittelte unser Betreuer die Vision und die aktuellen Anforderungen der Anwendung. Durch diese regelmäßigen Abstimmungen konnten Änderungen frühzeitig in den Entwicklungsprozess integriert werden, ohne bereits erstellte Komponenten komplett überarbeiten zu müssen. So konnten sowohl technische als auch gestalterische Anpassungen flexibel vorgenommen werden.

3.2 Projektorganisation

Folgende Tabelle zeigt eine IVM-Matrix, welche die Verantwortlichkeiten der einzelnen Projektmitglieder darstellt. Die Buchstaben in der Matrix stehen für unterschiedliche Rollen innerhalb der Aufgabenverteilung:

- **I** = Informiert: Das Mitglied wird über Fortschritte und Entscheidungen informiert.
- **V** = Verantwortlich: Das Mitglied trägt die Hauptverantwortung für die Aufgabe und stellt sicher, dass sie erfolgreich abgeschlossen wird.
- **M** = Mitwirkend: Das Mitglied beteiligt sich aktiv an der Umsetzung der Aufgabe, hat aber nicht die Hauptverantwortung.

Aufgabe / Aktivität	**Jan-Niclas Hartl**	**Maximilian Koch**	**Eric Reps**	**Luca Strobl**
Backendimplementation	M	M	V	I
Frontendimplementation	I	I	I	V
Datenbankverwaltung	M	V	M	M
Redis	V	I	I	I

Tabelle 1: IVM-Matrix der Projektmitglieder

3.3 Zeitplan und Meilensteine

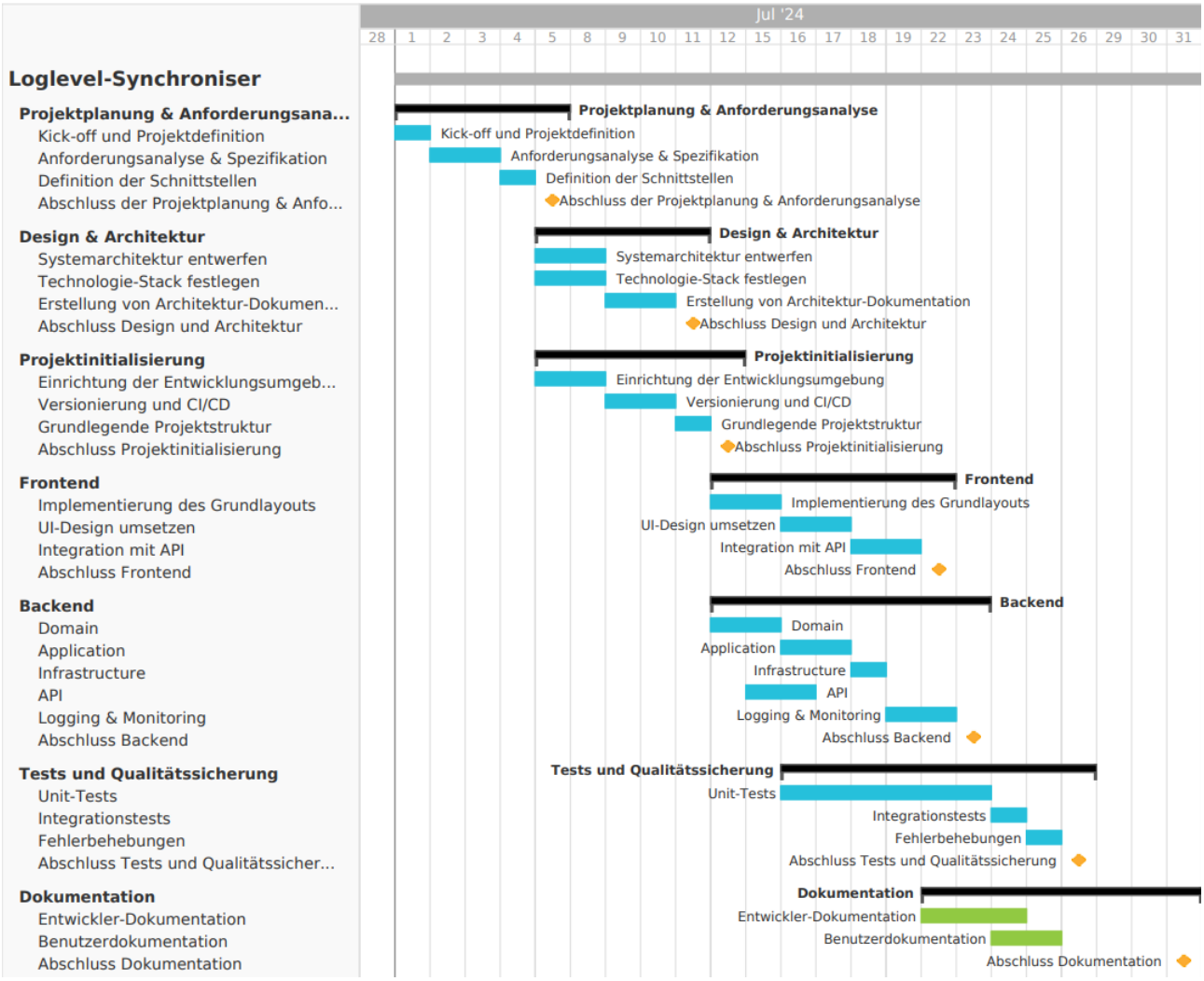


Abbildung 27: Projektplanung - Terminplan

In Abbildung 27 wird der Terminplan des Projekts ”Loglevel-Synchroniserïn Form eines Gantt-Diagramms dargestellt. Die einzelnen Aufgabenbereiche sind in einer hierarchischen Struktur gegliedert und als Balken entlang einer Zeitachse angeordnet. Dadurch werden die zeitliche Abfolge und die Abhängigkeiten zwischen den Aufgaben deutlich sichtbar.

Der Projektverlauf erstreckt sich über mehrere Wochen im Juli 2024, beginnend in Woche 27 und endend in Woche 31. Die Aufgaben sind in verschiedene Phasen unterteilt, darunter Projektplanung und Anforderungsanalyse, Design und Architektur, Projektinitialisierung, Frontend- und Backend-Entwicklung, Tests und Qualitätssicherung sowie Dokumentation.

Durch die Farbgebung der Balken werden unterschiedliche Aufgabenarten visualisiert. Blaue Balken repräsentieren wesentliche Implementierungsphasen, während grüne Balken den Bereich der Dokumentation abdecken. Schwarze Balken markieren übergeordnete Meilensteine, während diamantförmige Symbole wichtige Abschlussereignisse kennzeichnen. Die Verbindungen zwischen

den Vorgängen stellen logische Abhängigkeiten dar und zeigen, welche Aufgaben aufeinander aufbauen.

Die Struktur des Gantt-Diagramms ermöglicht eine übersichtliche Darstellung der geplanten Abläufe. Es wird ersichtlich, dass frühe Phasen wie Projektplanung, Design und Architektur und Projektinitialisierung entscheidend für nachfolgende Entwicklungen im Frontend und Backend sind. Die abschließenden Phasen umfassen Tests, Qualitätssicherung und Dokumentation, um eine vollständige und funktionale Umsetzung sicherzustellen.

3.3.1 Projektplannug und Anforderungsanalyse

Die Projektplanung und Anforderungsanalyse stellt eine essenzielle Phase in der Softwareentwicklung dar. In diesem Abschnitt werden die Grundlagen für das gesamte Projekt geschaffen, indem Ziele definiert, Anforderungen erhoben und technische Rahmenbedingungen festgelegt werden. Eine sorgfältige Planung und Analyse sind entscheidend, um spätere Anpassungen zu minimieren und eine effiziente Umsetzung sicherzustellen.

Kick-off und Projektdefinition

Der offizielle Projektstart dient dazu, eine gemeinsame Verständnisbasis für alle Beteiligten zu schaffen. In diesem Schritt werden die übergeordneten Ziele des Projekts definiert und wesentliche organisatorische sowie technische Rahmenbedingungen abgestimmt. Zudem erfolgt eine erste Identifikation der relevanten Stakeholder, die im weiteren Verlauf als Ansprechpartner für spezifische Anforderungen und Entscheidungen dienen.

Anforderungsanalyse und Spezifikation

Im Rahmen der Anforderungsanalyse werden funktionale und nicht-funktionale Anforderungen systematisch erfasst und dokumentiert. Die Erhebung der Anforderungen erfolgt durch detaillierte Gespräche mit dem zuständigen Firmenmitarbeiter, der die Applikation später nutzen wird. Diese direkte Kommunikation stellt sicher, dass die spezifischen Bedürfnisse und Erwartungen des Endnutzers präzise erfasst werden. Das Ergebnis dieser Phase ist eine detaillierte Anforderungsspezifikation, die als Grundlage für die Architektur- und Implementierungsentscheidungen dient.

Definition der Schnittstellen

Die Definition der Schnittstellen stellt sicher, dass die verschiedenen Systemkomponenten effizient miteinander kommunizieren können. In diesem Schritt werden die relevanten Kommunikationswege spezifiziert, einschließlich der auszutauschenden Daten, des verwendeten Datenformats sowie der eingesetzten Übertragungsprotokolle. Eine präzise Schnittstellenspezifikation trägt dazu bei, reibungslose Integration zwischen den Systemkomponenten sicherzustellen.

Abschluss der Projektplanung und Anforderungsanalyse

Der Abschluss dieser Phase umfasst eine umfassende Validierung und Freigabe der erstellten Spezifikationen. Dabei wird überprüft, ob alle Anforderungen klar definiert und mit den übergeordneten Projektzielen abgestimmt sind. Die abschließende Dokumentation dieser Ergebnisse bildet die Grundlage für die nachfolgenden Phasen der Softwareentwicklung und dient als Referenz für alle Projektbeteiligten.

3.3.2 Design und Architektur

Die Phase Design und Architektur bildet die Grundlage für die technische Umsetzung des Projekts. In diesem Schritt werden die Systemarchitektur entworfen, technologische Entscheidungen getroffen und eine strukturierte Dokumentation erstellt. Ziel ist es, eine robuste, skalierbare und wartbare Architektur zu konzipieren, die den zuvor definierten Anforderungen entspricht.

Systemarchitektur entwerfen

Die Systemarchitektur definiert die grundlegende Struktur der Anwendung und deren Komponenten. Dabei werden Aspekte wie die Verteilung der Funktionalitäten, die Interaktion zwischen Modulen sowie die zugrunde liegende Infrastruktur berücksichtigt. Die Architektur stellt sicher, dass das System sowohl performant als auch erweiterbar ist.

Technologie-Stack festlegen

Die Auswahl geeigneter Technologien hat einen entscheidenden Einfluss auf die Effizienz und Wartbarkeit der Anwendung. In diesem Schritt werden Programmiersprachen, Frameworks, Datenbanksysteme und weitere relevante Werkzeuge bestimmt. Die Entscheidung erfolgt unter Berücksichtigung von Faktoren wie Performance, Skalierbarkeit und langfristiger Unterstützung.

Erstellung von Architektur-Dokumentation

Eine strukturierte Dokumentation der Systemarchitektur dient als Referenz für Entwickler und andere Projektbeteiligte. Sie umfasst Diagramme, technische Spezifikationen und Richtlinien zur Implementierung. Die Dokumentation stellt sicher, dass Architekturentscheidungen nachvollziehbar bleiben und spätere Anpassungen effizient durchgeführt werden können.

Abschluss Design und Architektur

Vor dem Übergang zur Implementierung wird das Architekturkonzept final überprüft und freigegeben. Dabei wird geprüft, ob die entworfene Architektur alle Anforderungen erfüllt und eine solide Basis für die Entwicklung bildet. Die abschließende Dokumentation dieser Phase stellt sicher, dass alle relevanten Informationen für die nächste Projektstufe verfügbar sind.

3.3.3 Projektinitialisierung

Die Projektinitialisierung legt die technischen und organisatorischen Grundlagen für die Entwicklung. In dieser Phase wird die Entwicklungsumgebung eingerichtet, zentrale Prozesse wie Versionskontrolle und Continuous Integration/Continuous Deployment (CI/CD) etabliert und eine grundlegende Projektstruktur definiert. Ziel ist es, eine stabile und effiziente Arbeitsumgebung zu schaffen, die eine reibungslose Umsetzung des Projekts ermöglicht.

Einrichtung der Entwicklungsumgebung

Die Entwicklungsumgebung wird so konfiguriert, dass alle Teammitglieder unter gleichen Bedingungen arbeiten können. Dies umfasst die Installation notwendiger Software, die Einrichtung von Entwicklungs- und Testsystemen sowie die Konfiguration relevanter Entwicklungswerkzeuge.

Versionierung und CI/CD

Ein zentrales Element der Projektinitialisierung ist die Einrichtung der Versionskontrolle, um Änderungen am Quellcode nachvollziehbar zu machen. Zudem wird ein CI/CD-Prozess implementiert, der automatisierte Tests und die Bereitstellung der Anwendung ermöglicht. Dadurch wird eine hohe Codequalität sichergestellt und der Entwicklungsprozess beschleunigt.

Grundlegende Projektstruktur

Die Projektstruktur wird so definiert, dass eine klare Trennung von Modulen und Komponenten gewährleistet ist. Dies umfasst die Organisation des Quellcodes, Namenskonventionen sowie die Festlegung von Entwicklungsrichtlinien. Eine gut durchdachte Projektstruktur erleichtert die Zusammenarbeit und verbessert die Wartbarkeit des Systems.

Abschluss Projektinitialisierung

Nach der erfolgreichen Einrichtung aller notwendigen Systeme und Prozesse wird die Projektinitialisierung abgeschlossen. Dies stellt sicher, dass alle technischen Rahmenbedingungen geschaffen wurden und das Entwicklungsteam effizient mit der Implementierung beginnen kann.

3.3.4 Frontend

Das Frontend bildet die Benutzerschnittstelle der Anwendung und ist für die Interaktion mit dem Nutzer verantwortlich. In dieser Phase werden grundlegende Layouts implementiert, das UI-Design umgesetzt und die Integration mit dem Backend realisiert. Ziel ist es, eine benutzerfreundliche und funktionale Oberfläche zu entwickeln, die eine intuitive Bedienung der Applikation ermöglicht.

Implementierung des Grundlayouts

Die grundlegende Struktur der Benutzeroberfläche wird entwickelt, um eine konsistente und ansprechende Darstellung zu gewährleisten. Hierbei werden Navigationselemente, Seitengerüste und allgemeine Designrichtlinien umgesetzt, die als Basis für die weitere Entwicklung dienen.

UI-Design umsetzen

Das Benutzererlebnis (User Experience, UX) spielt eine zentrale Rolle bei der Gestaltung des Frontends. In diesem Schritt werden Designvorgaben umgesetzt, Farben, Typografie und Interaktionselemente definiert sowie responsives Verhalten sichergestellt. Dabei wird auf eine konsistente und ergonomische Gestaltung geachtet, um die Benutzerfreundlichkeit zu maximieren.

Integration mit API

Damit das Frontend mit dem Backend kommunizieren kann, wird eine Anbindung an die definierten Schnittstellen (APIs) realisiert. Hierbei werden Mechanismen zur Datenübertragung

implementiert, API-Endpunkte angebunden und Datenstrukturen entsprechend den Spezifikationen verarbeitet. Eine effiziente und sichere Kommunikation zwischen Frontend und Backend ist essenziell für die Gesamtfunktionalität der Applikation.

Abschluss Frontend

Nachdem alle wesentlichen Komponenten des Frontends umgesetzt und erfolgreich getestet wurden, wird diese Phase abgeschlossen. Dies stellt sicher, dass die Benutzeroberfläche den Anforderungen entspricht und eine stabile Basis für weitere Optimierungen und Erweiterungen bietet.

3.3.5 Backend

Das Backend stellt die zentrale Geschäftslogik und Datenverarbeitung der Anwendung bereit. Es verwaltet die Datenpersistenz, verarbeitet Anfragen des Frontends und gewährleistet die Sicherheit sowie Skalierbarkeit des Systems. In dieser Phase werden die Kernkomponenten des Backends implementiert, darunter die Domänenschicht, Anwendungslogik, Infrastruktur und Schnittstellen.

Domain

Die Domänenschicht bildet die fachliche Basis der Anwendung und definiert die zentralen Geschäftsobjekte sowie deren Beziehungen. Sie stellt sicher, dass alle relevanten Geschäftsregeln korrekt modelliert und implementiert sind, sodass die Anwendung die gewünschten Funktionen zuverlässig ausführt.

Application

In der Anwendungsschicht wird die Geschäftslogik implementiert. Hierbei werden Anfragen verarbeitet, Geschäftsregeln angewendet und die Interaktion zwischen verschiedenen Systemkomponenten gesteuert. Diese Schicht trennt die fachliche Logik von der technischen Umsetzung und ermöglicht so eine saubere, modularisierte Architektur.

Infrastructure

Die Infrastrukturschicht umfasst die technischen Komponenten, die für den Betrieb des Backends erforderlich sind. Dazu gehören Datenbankanbindungen, externe Dienste, Authentifizierungsme-

chanismen und Speicherlösungen. Eine effiziente und robuste Infrastruktur ist essenziell für die Performance und Skalierbarkeit des Systems.

API

Die Programmierschnittstelle (API) ermöglicht die Kommunikation zwischen Frontend und Backend. In diesem Schritt werden Endpunkte definiert, Sicherheitsmechanismen wie Authentifizierung und Autorisierung implementiert und Datenformate standardisiert. Eine gut strukturierte API gewährleistet eine reibungslose Integration und erleichtert zukünftige Erweiterungen.

Logging und Monitoring

Um eine zuverlässige Überwachung und Fehleranalyse zu ermöglichen, werden Logging- und Monitoring-Mechanismen implementiert. Dies umfasst das Erfassen von Systemereignissen, die Analyse von Leistungsmetriken und das automatische Erkennen von Fehlerzuständen. Ein effektives Monitoring stellt sicher, dass Probleme frühzeitig erkannt und behoben werden können.

Abschluss Backend

Nach der erfolgreichen Implementierung und Überprüfung aller Backend-Komponenten wird diese Phase abgeschlossen. Dies gewährleistet, dass die Geschäftslogik stabil und zuverlässig funktioniert, eine sichere Kommunikation mit anderen Systemen möglich ist und eine solide Grundlage für den weiteren Entwicklungsprozess geschaffen wurde.

3.3.6 Tests und Qualitätssicherung

Die Tests und Qualitätssicherung stellen sicher, dass die entwickelte Software stabil, zuverlässig und fehlerfrei funktioniert. Dieser Meilenstein umfasst die Implementierung von Tests, die Durchführung von Integrationstests und die Behebung identifizierter Fehler. Ziel ist es, die Qualität der Anwendung zu gewährleisten und potenzielle Probleme frühzeitig zu erkennen und zu beheben.

Unit-Tests

Unit-Tests sind grundlegende Tests, die einzelne Komponenten oder Funktionen der Anwendung isoliert prüfen. In dieser Phase werden alle wichtigen Geschäftslogik-Teile, wie Berechnungen

und Datenverarbeitungsfunktionen, durch Unit-Tests abgedeckt. Dadurch wird sichergestellt, dass jede einzelne Einheit der Anwendung unabhängig von anderen korrekt arbeitet und erwartungsgemäß funktioniert.

Integrationstests

Integrationstests überprüfen, wie gut die einzelnen Komponenten der Anwendung miteinander interagieren. In dieser Phase werden die Schnittstellen zwischen verschiedenen Schichten des Systems getestet, beispielsweise die Kommunikation zwischen der Datenbank und der Anwendung oder die Interaktion zwischen Frontend und Backend. Ziel ist es, potenzielle Fehler in der Zusammenarbeit der Module zu identifizieren und die ordnungsgemäße Funktionsweise des Gesamtsystems zu gewährleisten.

Fehlerbehebungen

Während der Tests werden Fehler und Unregelmäßigkeiten identifiziert, die die Funktionsweise der Anwendung beeinträchtigen können. In dieser Phase werden alle gefundenen Fehler systematisch analysiert, priorisiert und behoben. Dies umfasst sowohl technische Fehler als auch Probleme mit der Benutzeroberfläche oder der Anwendungslogik. Durch die Fehlerbehebung wird die Qualität und Stabilität der Anwendung weiter verbessert.

Abschluss Tests und Qualitätssicherung

Nach der erfolgreichen Durchführung aller Tests und der Behebung der identifizierten Fehler wird die Phase der Tests und Qualitätssicherung abgeschlossen. Dies garantiert, dass die Anwendung stabil, sicher und frei von kritischen Fehlern ist. Die Qualität der Software wird durch umfassende Tests sichergestellt, sodass das System zuverlässig in Produktion gehen kann und zukünftige Erweiterungen problemlos möglich sind.

3.3.7 Dokumentation

Die Dokumentation dient der klaren und verständlichen Aufzeichnung aller relevanten Aspekte des Projekts. Sie stellt sicher, dass sowohl Entwickler als auch Endbenutzer die Anwendung problemlos nutzen und weiterentwickeln können. In dieser Phase werden die Entwicklerdokumentation sowie die Benutzerdokumentation erstellt, um eine vollständige und nachvollziehbare Aufzeichnung des Projekts zu gewährleisten.

Entwickler Dokumentation

Die Entwicklerdokumentation beschreibt die technischen Details der Anwendung, einschließlich der Architektur, der verwendeten Technologien und der Struktur des Codes. Hier werden alle wichtigen Komponenten, Schnittstellen und Funktionsweisen der Software detailliert dokumentiert, sodass zukünftige Entwickler die Anwendung leicht verstehen und weiterentwickeln können. Diese Dokumentation umfasst unter anderem Installationsanleitungen, Codebeispiele, Datenbankstruktur sowie Hinweise zur Erweiterung und Wartung des Systems.

Benutzerdokumentation

Die Benutzerdokumentation richtet sich an die Endanwender der Anwendung und erklärt die Bedienung sowie die wichtigsten Funktionen des Systems. In dieser Phase werden alle relevanten Schritte und Anleitungen beschrieben, um den Benutzern eine einfache und intuitive Nutzung der Anwendung zu ermöglichen. Dies umfasst Installationshinweise, eine detaillierte Beschreibung der Benutzeroberfläche, häufig gestellte Fragen (FAQs) sowie Problemlösungsstrategien für häufig auftretende Probleme.

Abschluss Dokumentation

Nach der Fertigstellung der Entwickler- und Benutzerdokumentation wird die Phase der Dokumentation abgeschlossen. Dies stellt sicher, dass alle notwendigen Informationen sowohl für die Weiterentwicklung durch Entwickler als auch für die Nutzung durch Endanwender vollständig und korrekt dokumentiert sind. Eine gut strukturierte Dokumentation bildet die Grundlage für eine langfristige Wartung und Weiterentwicklung der Anwendung sowie eine einfache Handhabung durch die Benutzer.

4 Design

Dieses Kapitel beschreibt das Design des LogLevel-Synchronisers und legt dar, wie die Anforderungen in eine klare, strukturierte Architektur umgesetzt wurden. Dabei wird insbesondere auf die Anwendung der Prinzipien des Domain-Driven Designs (DDD) und der schichtweisen Architektur eingegangen, die eine klare Trennung zwischen Geschäftslogik, Anwendungslogik, Infrastruktur und API sicherstellt.

4.1 Architektur des Loglevel-Synchronisers

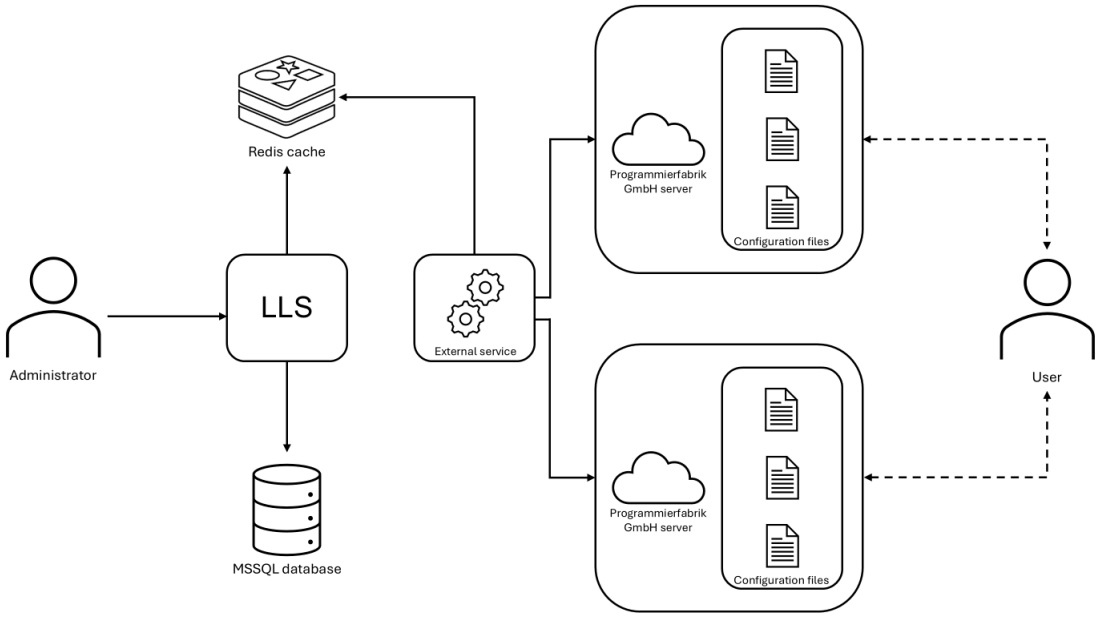


Abbildung 28: Loglevel-Synchroniser - Architektur

Die vorliegende Grafik (siehe Abbildung 28) stellt die Architektur einer zentralisierten Log-Level-Verwaltung dar, welche die administrative Steuerung und Synchronisation der Logging-Konfigurationen auf mehreren Servern der Programmierfabrik GmbH ermöglicht. Die Architektur zielt darauf ab, die bisher manuelle und fehleranfällige Konfiguration von Log-Leveln zu automatisieren, um eine effizientere Verwaltung und eine konsistente Anwendung der Einstellungen zu gewährleisten.

Komponenten und deren Interaktion

Administrator und Loglevel-Synchroniser (LLS)

Der Administrator interagiert direkt mit dem Loglevel-Synchroniser, um Änderungen an den Log-Leveln der verschiedenen Sites vorzunehmen. Diese Konfigurationen basieren auf den Informationen, die der LLS aus der angeschlossenen MSSQL-Datenbank bezieht. Die Datenbank enthält spezifische Informationen zu den vorhandenen Sites sowie den jeweils zugeordneten Logging-Bibliotheken.

Konfigurationsvorgang

Wenn der Administrator beispielsweise das Log-Level einer Site wie gw.stammportal.at von Error auf Debug ändert, wird diese Konfigurationsanpassung an das LLS-Backend weitergeleitet. Das Backend verarbeitet die Anfrage und speichert den entsprechenden Befehl im Redis-Cache.

Falls der Administrator zudem eine Reset-Time festlegt, wird dies ebenfalls in den generierten Befehl integriert. Die Reset-Time definiert einen Zeitpunkt, zu dem das Log-Level automatisch auf den vorherigen Wert zurückgesetzt wird, um zu verhindern, dass eine zu lange Aktivierung von detaillierten Log-Leveln unnötig Speicherplatz beansprucht.

Redis-Cache und externer Service

Der Redis-Cache spielt eine zentrale Rolle in der Synchronisation der Log-Level. Ein externer Service überwacht kontinuierlich den Redis-Cache auf Veränderungen. Sobald eine neue Konfigurationsanpassung erkannt wird, liest dieser Dienst die aktualisierten Parameter aus und wendet die Änderungen an den zu verwaltenden Servern an.

Dieser Ansatz gewährleistet eine skalierbare Infrastruktur: Falls die Programmierfabrik GmbH die Anzahl der verwalteten Server von zwei auf vier erweitert, muss lediglich der externe Service angepasst werden, ohne dass grundlegende Änderungen an der Gesamtarchitektur erforderlich sind.

Server und Konfigurationsdateien

Jeder verwaltete Server enthält spezifische Konfigurationsdateien, die für die jeweilige Site definieren, welche Log-Level gesetzt werden können. Sobald der externe Service eine Anpassung

vornimmt, werden diese Konfigurationsdateien entsprechend der Anweisungen aus dem Redis-Cache aktualisiert.

Anwendung der neuen Log-Level

Nach erfolgreicher Umsetzung der Änderungen durch den externen Service sind die neuen Log-Level auf den entsprechenden Sites aktiv. In dem obigen Beispiel bedeutet dies, dass nun alle Debug-Informationen in die Logging-Dateien geschrieben werden, bis entweder eine neue Änderung durch den Administrator vorgenommen wird oder die Reset-Time die Einstellungen auf den vorherigen Zustand zurücksetzt.

Vorteile der Architektur

Diese Lösung bringt mehrere Vorteile mit sich. Zum einen wird die bisher manuelle Konfiguration durch eine zentrale Steuerung automatisiert, was nicht nur den administrativen Aufwand deutlich reduziert, sondern auch potenzielle Fehlerquellen minimiert. Darüber hinaus ist die Architektur auf Skalierbarkeit ausgelegt: Neue Server lassen sich unkompliziert in die bestehende Infrastruktur integrieren, indem lediglich der externe Service angepasst wird.

Ein weiterer Vorteil liegt in der erhöhten Flexibilität. Die Möglichkeit, eine Reset-Time zu definieren, erlaubt es, Log-Level automatisch zurückzusetzen, wodurch Speicherplatz und Systemressourcen effizient genutzt werden. Gleichzeitig wird eine konsistente und synchronisierte Konfiguration über alle Server hinweg sichergestellt, sodass eine einheitliche Log-Level-Strategie jederzeit gewährleistet ist.

Insgesamt ermöglicht diese Architektur eine effiziente und zuverlässige Verwaltung der Logging-Konfigurationen innerhalb der Programmierfabrik GmbH, was sowohl die Betriebssicherheit als auch die Wartungsfreundlichkeit erheblich steigert.

4.2 Datenbankschema

Das Datenbankschema besteht aus mehreren Tabellen, die unterschiedliche Aspekte des Systems modellieren. Im Folgenden wird jede Tabelle und ihre Bedeutung im Detail beschrieben.

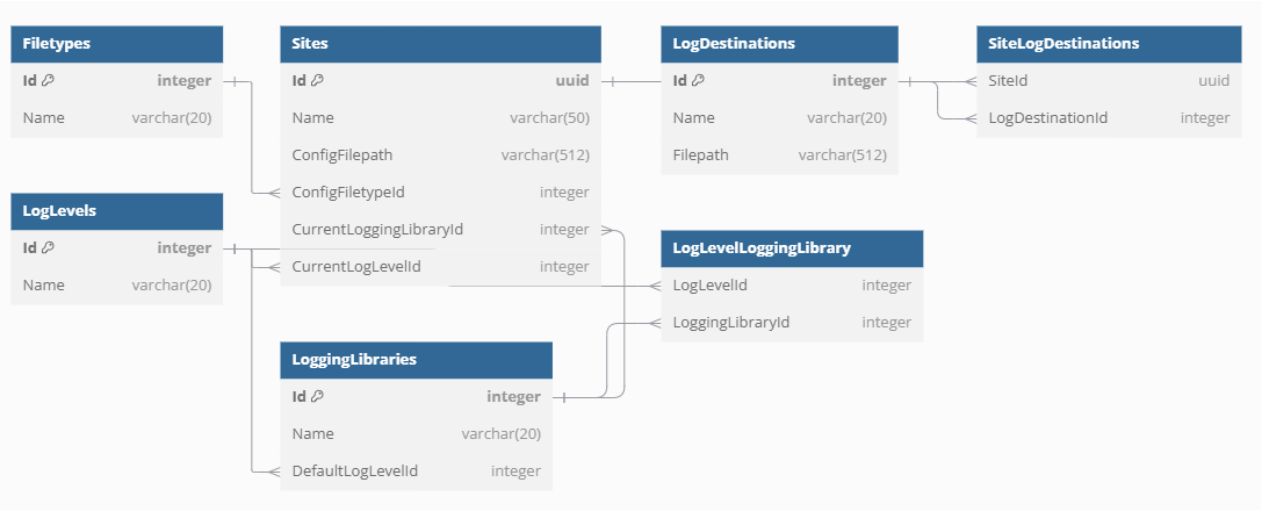


Abbildung 29: LogLevel-Synchroniser - Datenbankschema

Filetypes

Die Tabelle Filetypes speichert die Dateitypen, die für die Konfiguration der Sites verwendet werden. Jede Zeile in der Tabelle stellt einen Dateityp dar. Ein Filetype gibt an, welchen Typ die Konfigurationsdatei einer Site hat. Jede Site hat einen Pfad zu ihrer Konfigurationsdatei, in der unter anderem das LogLevel festgelegt wird. Diese Datei kann verschiedene Formate haben, zum Beispiel JSON oder XML. Es ist wichtig, den Typ dieser Datei zu kennen, weil unterschiedliche Dateiformate auf verschiedene Weise verarbeitet werden müssen. JSON-Dateien sind zum Beispiel strukturiert und können leicht von Programmen gelesen werden, während XML-Dateien eine andere Struktur haben. Daher hilft die Information über den Dateityp dabei, die Datei richtig zu interpretieren und weiterzuverarbeiten.

LogLevels

Ein LogLevel gibt an, wie detailliert oder wichtig eine Log-Nachricht ist. In einem Logging-System werden Nachrichten in verschiedenen Schweregraden erfasst, um zu entscheiden, welche Informationen für den Entwickler oder Administrator wichtig sind. Typische Log-Level umfassen:

Diese Log-Level werden von unterschiedlichen Logging-Bibliotheken gesetzt. Jede Bibliothek kann so konfiguriert werden, dass sie bestimmte Log-Level unterstützt, je nachdem, welche Informationen der Entwickler für wichtig hält. Die Tabelle LogLevels speichert diese verschiedenen Log-Level und ordnet ihnen eine eindeutige ID zu. So kann das System genau bestimmen, welche Art von Informationen in die Logs geschrieben werden, abhängig von der Schwere des Ereignisses.

Level	Beschreibung
`Debug`	Sehr detaillierte Informationen, die hauptsächlich zur Fehlerbehebung während der Entwicklung verwendet werden.
`Info`	Allgemeine Informationen über den normalen Betrieb eines Systems.
`Error`	Fehler, die während der Ausführung auftreten, aber das System weiterhin funktionieren lässt.
`Warn`	Warnungen, die auf potenzielle Probleme hinweisen, die jedoch keine sofortige Handlung erfordern.
`Off`	Das Logging ist vollständig deaktiviert.
`Fatal`	Kritische Fehler, die das System zum Absturz bringen oder die Ausführung unmöglich machen.

Tabelle 2: Logging-Level und ihre Beschreibungen

Sites

Die Tabelle Sites speichert Informationen zu den Seiten, die von der Programmierfabrik GmbH verwaltet werden. Eine Site ist dabei eine Webseite, wie zum Beispiel gw.stammportal.at. Jede Site in dieser Tabelle hat eine eindeutige ID (eine UUID), die sie von anderen Seiten unterscheidet. Neben dem Namen der Site, der auch in der Tabelle gespeichert wird, hat jede Seite einen Pfad zu einer Konfigurationsdatei. Diese Konfigurationsdatei ist eine spezielle Datei, in der das LogLevel für die Site festgelegt wird. Das LogLevel gibt an, wie detailliert oder schwerwiegend die Log-Nachrichten sein sollen (zum Beispiel „Info“, „Error“ oder „Fatal“).

Diese Konfigurationsdatei hat, wie bereits bei der Tabelle Filetypes erklärt, einen bestimmten Dateityp (z. B. JSON oder XML), der in der Tabelle Filetypes gespeichert wird. Der Dateityp ist wichtig, weil die Konfigurationsdatei je nach Format auf unterschiedliche Weise verarbeitet und gelesen werden muss.

Die Informationen aus der Konfigurationsdatei sind für den externen Service der Programmierfabrik GmbH sehr wichtig, weil dieser Service diese Daten benötigt, um Änderungen an der Konfiguration der Site vorzunehmen. Das bedeutet, dass der Service in der Lage ist, das LogLevel und andere Konfigurationsparameter anzupassen, je nach den Anforderungen der Site.

Zusätzlich speichert die Tabelle Sites das LogLevel, das aktuell für die Site verwendet wird. Dies wird zusammen mit der Logging-Bibliothek gespeichert, die zum Aufzeichnen der Logs verwendet wird, zum Beispiel Serilog. Serilog ist eine bekannte Logging-Bibliothek, die in vielen Anwendungen genutzt wird, um Logs zu generieren und zu speichern.

LoggingLibraries

Die Tabelle LoggingLibraries speichert Informationen über verschiedene Logging-Bibliotheken. Eine Logging-Bibliothek ist eine spezielle Software-Komponente, die dazu verwendet wird, Log-Nachrichten in einer Anwendung zu verwalten und zu speichern. Beispiele für solche Bibliotheken sind Serilog, log4net oder NLog.

Jede Logging-Bibliothek kann unterschiedlich funktionieren und bietet verschiedene Möglichkeiten, Logs zu erfassen, zu formatieren und zu speichern. Manche Bibliotheken unterstützen zum Beispiel strukturierte Logs in JSON-Format, während andere einfacher gehalten sind und reine Textdateien schreiben.

Die Tabelle LoggingLibraries enthält eine eindeutige ID und den Namen der jeweiligen Bibliothek. Zusätzlich speichert sie ein Standard-LogLevel, das immer dann verwendet wird, wenn für eine Site kein spezifisches LogLevel festgelegt wurde. Dieses Standard-LogLevel wird in der Tabelle LogLevels gespeichert und ist über die Spalte DefaultLogLevelId mit der Tabelle LogLevels verknüpft.

Jede Site nutzt genau eine Logging-Bibliothek, die in der Tabelle Sites gespeichert wird. Das bedeutet, dass für jede Site festgelegt wird, mit welcher Logging-Bibliothek sie arbeitet. So kann eine Site beispielsweise Serilog nutzen, während eine andere mit log4net arbeitet.

Außerdem gibt es eine weitere Verbindung zur Tabelle LogLevels über die Tabelle LogLevelLoggingLibrary. Diese sorgt dafür, dass jede Logging-Bibliothek genau festlegt, welche Log-Level sie unterstützt. Das ist wichtig, weil nicht jede Bibliothek dieselben Log-Level hat oder gleich damit umgeht. Manche Logging-Bibliotheken erlauben beispielsweise nur die Stufen „Info“, „Warn“ und „Error“, während andere feinere Abstufungen wie „Trace“ oder „Fatal“ anbieten.

LogDestinations

Die Tabelle LogDestinations speichert Informationen darüber, wohin die Logs einer Site geschrieben werden. In einem Logging-System ist es wichtig, dass die erfassten Log-Nachrichten an den richtigen Ort gespeichert werden, damit sie später analysiert oder überprüft werden können.

Eine LogDestination ist ein Speicherort für Logs. Das kann zum Beispiel eine Datei auf einem Server sein, eine zentrale Datenbank oder ein externes Logging-System. Jede LogDestination hat eine eindeutige ID, einen Namen und einen Dateipfad. Der Dateipfad gibt an, wo genau die Logs gespeichert werden.

Ein wichtiger Punkt ist, dass eine Site mehrere LogDestinations haben kann. Das bedeutet, dass die Logs einer Site an verschiedene Orte gleichzeitig geschrieben werden können. Zum Beispiel könnte eine Site ihre Logs sowohl in eine Datei als auch in eine Datenbank speichern. Das kann nützlich sein, wenn die Logs an mehreren Stellen benötigt werden, zum Beispiel für Fehleranalysen oder zur Langzeitarchivierung.

Um die Verbindung zwischen Sites und LogDestinations darzustellen, gibt es eine zusätzliche Tabelle namens SiteLogDestinations. Diese sorgt dafür, dass eine Site mit mehreren LogDestinations verknüpft werden kann und umgekehrt eine LogDestination auch Logs von mehreren Sites aufnehmen kann.

LogLevelLoggingLibrary

Die Tabelle LogLevelLoggingLibrary ist eine sogenannte Zwischentabelle. Sie stellt eine Verbindung zwischen den Tabellen LogLevels und LoggingLibraries her.

Jede Logging-Bibliothek kann unterschiedliche Log-Level unterstützen. Manche haben nur grundlegende Stufen wie „Info“, „Warn“ und „Error“, während andere noch detailliertere Abstufungen wie „Trace“ oder „Fatal“ haben. Diese Unterschiede müssen in der Datenbank abgebildet werden, damit klar ist, welche Log-Level in welcher Bibliothek verwendet werden können.

Diese Tabelle besteht aus zwei Spalten: LogLevelId und LoggingLibraryId. Die Spalte LogLevelId verweist auf einen Eintrag in der Tabelle LogLevels, während LoggingLibraryId auf eine Logging-Bibliothek in der Tabelle LoggingLibraries zeigt.

Das bedeutet, dass in dieser Tabelle genau festgelegt wird, welche Log-Level von welcher Logging-Bibliothek unterstützt werden. So kann zum Beispiel Serilog alle Log-Level haben, während eine andere Bibliothek nur einige wenige unterstützt.

Ohne diese Zwischentabelle wäre es schwieriger zu verwalten, welche Log-Level von welcher Bibliothek verwendet werden dürfen. Durch diese Lösung kann das System flexibel bleiben, falls neue Logging-Bibliotheken oder neue Log-Level hinzugefügt werden müssen.

SiteLogDestinations

Die Tabelle SiteLogDestinations ist eine weitere Zwischentabelle. Sie verbindet die Tabellen Sites und LogDestinations miteinander.

Der Grund für diese Verbindung ist, dass eine Site ihre Logs an mehrere verschiedene Ziele schreiben kann. Gleichzeitig kann eine LogDestination auch Logs von mehreren Sites aufnehmen. Um das möglich zu machen, gibt es keine direkte Verbindung zwischen Sites und LogDestinations, sondern diese Zwischentabelle.

Die Tabelle hat zwei Spalten: SiteId und LogDestinationId. SiteId verweist auf eine Site aus der Tabelle Sites, während LogDestinationId auf einen Eintrag in der Tabelle LogDestinations zeigt.

Durch diese Verbindung kann eine Site ihre Logs gleichzeitig in mehrere Speicherorte schreiben, z. B. in eine Datei und zusätzlich in eine zentrale Datenbank. Gleichzeitig kann eine einzelne LogDestination auch Logs von mehreren Sites enthalten. Das kann nützlich sein, wenn zum Beispiel eine zentrale Datenbank alle Logs von verschiedenen Sites speichert.

Ohne diese Zwischentabelle wäre es nicht möglich, dass eine Site mehrere LogDestinations hat oder eine LogDestination von mehreren Sites genutzt wird. Diese Lösung sorgt also für mehr Flexibilität im System.

4.3 Domain-Driven Design

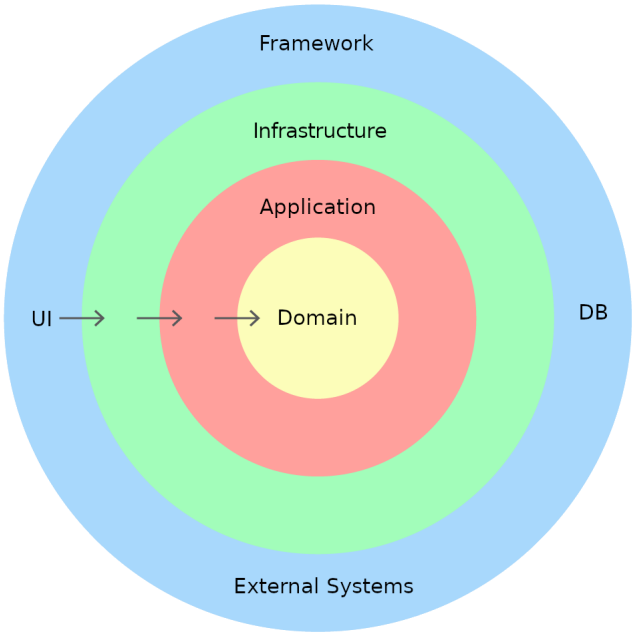


Abbildung 30: Domain-Driven Design (DDD) [49]

Unsere Anwendung orientiert sich am Ansatz des Domain-Driven Design (DDD), der die Geschäftsdomäne in den Mittelpunkt stellt und somit die realen fachlichen Anforderungen präzise

abbildet. Im Fokus steht dabei nicht primär eine strikte Trennung von Geschäftslogik und technischer Implementierung, sondern die modellgetriebene Ausrichtung an den Geschäftsprozessen und -regeln . Nichtsdestotrotz resultiert dieser Ansatz in einer klar strukturierten Architektur, die die Modularität, Testbarkeit und Wartbarkeit des Systems fördert. Die isolierte Betrachtung einzelner Schichten ermöglicht es, die Geschäftslogik unabhängig von infrastrukturellen Anpassungen zu validieren und technische Änderungen gezielt in der jeweiligen Schicht vorzunehmen.

Aufbau des Designs

Wie in Abbildung 30 zu sehen ist, besteht die Architektur aus mehreren Schichten. Im Zentrum steht die Domain, die das Herzstück der Anwendung bildet. Sie enthält sämtliche Geschäftsregeln, Domänenlogik und Entitäten. Wichtig ist hierbei, dass diese Schicht vollkommen unabhängig von der technischen Implementierung bleibt. Dadurch wird sichergestellt, dass die Geschäftslogik nicht von spezifischen Frameworks oder Datenbanktechnologien beeinflusst wird.

Domain

Die Domain-Schicht bildet das Herzstück der Anwendung. Hier sind alle zentralen Geschäftsregeln definiert. Diese Schicht enthält die Entitäten, Wertobjekte und die Domänenlogik, die unabhängig von der technischen Implementierung sein müssen. Da die Domain keine Abhängigkeiten zu anderen Schichten haben sollte, bleibt sie stabil und kann unabhängig von Frameworks oder Datenbanken verwendet werden.

Application

Die Application-Schicht steuert die Abläufe innerhalb der Anwendung und stellt sicher, dass die definierten Geschäftsprozesse korrekt ausgeführt werden. Sie enthält sogenannte Use Cases oder Services, die Methoden der Domain aufrufen, um die Anwendungsfälle umzusetzen. Diese Schicht selbst enthält keine Geschäftslogik, sondern koordiniert lediglich die Interaktionen zwischen den Schichten.

[49] https://cdn.hibit.dev/images/posts/2021/ddd_layers.png?fmt=webp

Infrastructure

In der Infrastructure-Schicht befindet sich die technische Umsetzung der Anwendung. Hier werden Datenbankzugriffe, externe Schnittstellen und andere Infrastrukturkomponenten implementiert. In unserem Fall nutzen wir das Entity Framework Core, um den Zugriff auf die Datenbank zu verwalten. Diese Schicht ist von der Domain getrennt, sodass Änderungen an der Infrastruktur keine Auswirkungen auf die Geschäftslogik haben.

API

Da unsere Anwendung als Web-API entwickelt wurde, gibt es eine zusätzliche API-Schicht. Diese enthält die Controller, die HTTP-Anfragen entgegennehmen und die entsprechenden Prozesse in der Application-Schicht auslösen. Die API-Schicht dient als Schnittstelle zwischen der Anwendung und externen Clients, sodass verschiedene Systeme über REST- oder andere Web-Protokolle mit unserer Anwendung kommunizieren können.

Framework

Die äußere Schicht des Modells umfasst das Framework und andere externe Systeme. Hierzu gehören Betriebssysteme, Datenbanken oder Cloud-Dienste, mit denen die Anwendung interagiert. Diese äußere Schicht stellt die Umgebung bereit, in der die Anwendung betrieben wird.

4.4 Schnittstellendefinition

Tabelle 3 fasst die zentralen API-Endpunkte des Backends zusammen. Sie bietet eine übersichtliche Darstellung, in der jedem Endpunkt eine präzise Funktion zugeordnet wird – von der Abfrage einer vollständigen Seitenliste bis zur Anpassung spezifischer Log-Level.

Endpunkt	**Beschreibung**
GET /api/sites	Liefert eine Liste aller Seiten
GET /api/sites/siteId/logLevels	Liefert eine Liste aller Log-Level einer Seite
PUT /api/sites/siteId/updateLogLevel	Verändert das Log-Level einer Seite

Tabelle 3: Schnittstellendefinition

4.5 Client-Server Kommunikation

Die Kommunikation zwischen dem Backend und dem Frontend erfolgt über das Protokoll Representational State Transfer (REST), vgl. Abschnitt 2.1.6. Dabei stehen die in Abschnitt 4.4 definierten Schnittstellen zur Verfügung. Das Backend und das Frontend interagieren über eine klar definierte REST-API, welche eine effiziente und standardisierte Datenübertragung gewährleistet.

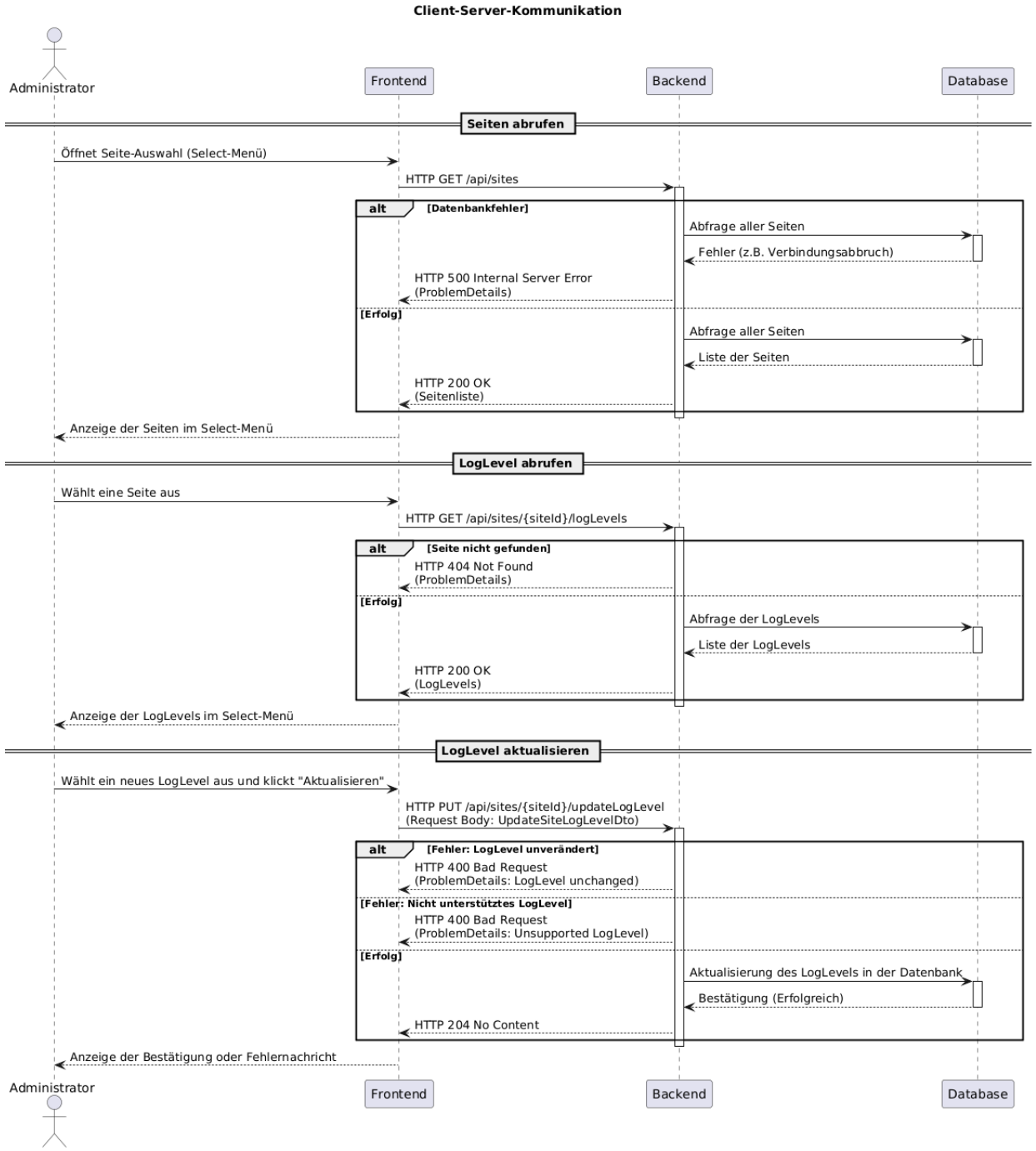


Abbildung 31: Client-Server Kommunikation

Wie in Abbildung 31 dargestellt, startet der Ablauf damit, dass der Administrator im Frontend die Seiten-Auswahl öffnet. Das Frontend ruft daraufhin per HTTP GET /api/sites alle verfügbaren Seiten beim Backend ab. Im Fehlerfall (z. B. Datenbankfehler) wird ein HTTP 500 Internal Server Error zurückgegeben, bei Erfolg hingegen HTTP 200 OK mit einer Liste der Seiten.

Anschließend kann der Administrator eine bestimmte Seite auswählen, um die dazugehörigen LogLevels abzurufen. Hierzu wird eine HTTP GET /api/sites/{siteId}/logLevels-Anfrage an das Backend gestellt. Wird eine ungültige siteId übermittelt, so folgt ein HTTP 404 Not Found. Ist die Seite jedoch vorhanden, antwortet das System mit HTTP 200 OK und einer Liste von LogLevels.

Abschließend kann der Administrator ein neues LogLevel auswählen und dieses über HTTP PUT /api/sites/{siteId}/updateLogLevel (Request Body: UpdateSiteLogLevelDto) aktualisieren. Tritt dabei beispielsweise der Fehler auf, dass dasselbe LogLevel erneut übertragen wird oder ein nicht unterstütztes LogLevel übergeben wird, so gibt das Backend einen HTTP 400 Bad Request zurück. Im Erfolgsfall wird die Aktualisierung durch HTTP 204 No Content bestätigt, was signalisiert, dass der Vorgang ausgeführt wurde, ohne zusätzliche Inhalte zurückzuliefern.

4.6 Aktivitätsdiagramm

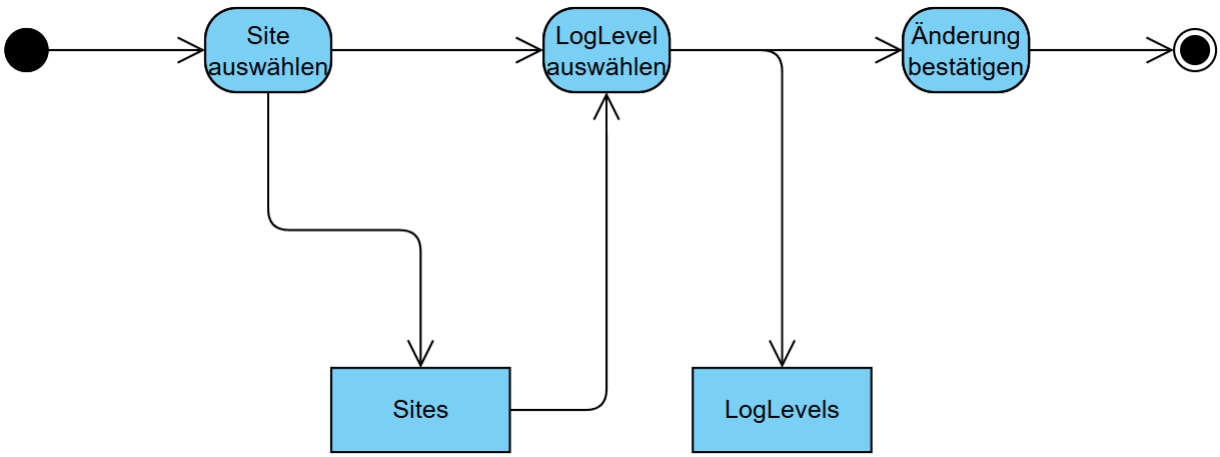


Abbildung 32: LogLevel-Synchroniser - Aktivitätsdiagramm

Das Aktivitätsdiagramm in Abbildung 32 beschreibt den Ablauf zur zentralisierten Änderung eines Log-Levels für eine gehostete Site über die entwickelte Webanwendung. Der Prozess stellt sicher, dass nur für die jeweilige Site gültige Log-Level zur Auswahl stehen und die Änderung konsistent auf beiden Servern umgesetzt wird.

Der Ablauf beginnt mit der Auswahl einer Site durch den Administrator. Diese Auswahl wird im System gespeichert und bestimmt, welche Log-Level im nächsten Schritt zur Verfügung stehen. Nur die für die gewählte Site zulässigen Log-Level können ausgewählt werden.

Anschließend wählt der Administrator ein Log-Level aus den verfügbaren Optionen.

Nach der Bestätigung der Änderung mit „Änderung bestätigen“ wird die neue Konfiguration gespeichert und synchronisiert. Dadurch wird das Log-Level für die Site auf beiden Servern aktualisiert.

4.7 Klassendiagramm

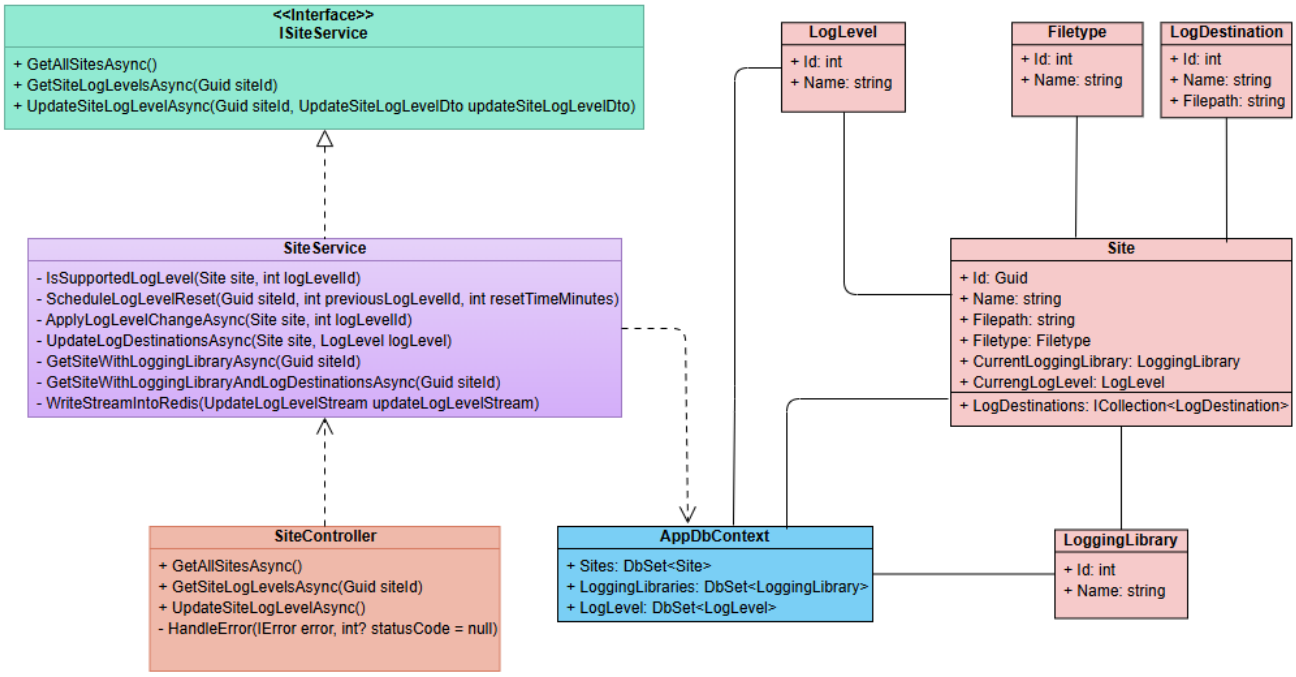


Abbildung 33: LogLevel-Synchroniser - Klassendiagramm

Abbildung 33 veranschaulicht die Struktur der Implementierung des LogLevel-Synchronisers. Die Architektur der Anwendung ist in drei Hauptschichten unterte Im Folgenden werden die einzelnen Klassen und deren Beziehungen detailliert beschrieben.

Klasse	**Verweis**
Filetype	siehe Listing 9
LogDestination	siehe Listing 10
LoggingLibrary	siehe Listing 11
LogLevel	siehe Listing 12
Site	siehe Listing 13
ErrorMessages	siehe Listing 14
RedisStreamConfig	siehe Listing 22
UpdateLogLevelStream	siehe Listing 19
AppDbContext	siehe Listing 5.1.3
ISiteService	siehe Listing 17
SiteController	siehe Listing 63

Tabelle 4: Übersicht der Klassen und deren Erklärungen/Verweise

5 Implementierung

Die in diesem Kapitel beschriebene praktische Umsetzung des LogLevel-Synchronisers basiert auf den in Abschnitt 4.3 erläuterten Konzepten des Domain-Driven Designs (DDD), das eine klare Trennung zwischen Geschäftslogik, technischer Infrastruktur und der API ermöglicht.

5.1 Backend-Implementierung

5.1.1 Domänenschicht

In diesem Abschnitt wird erläutert, wie das zuvor beschriebene Datenbankschema (siehe Abschnitt 4.2) in der Domänenschicht umgesetzt wurde.

```
1  public class Filetype
2  {
3      public int Id { get; set; }
4      public string Name { get; set; }
5  }
```

Listing 9: C# Entität - Filetype

Listing 9 zeigt die Definition der Filetype-Entität. Ein Filetyp gibt an, welchen Typ die Konfigurationsdatei einer Site besitzt – beispielsweise JSON oder XML. Wie bereits im Kapitel 4.2 beschrieben, ist der Dateityp entscheidend für die Verarbeitung der Datei, da unterschiedliche Formate auf unterschiedliche Weise interpretiert werden müssen.

```
1  public class LogDestination
2  {
3      public int Id { get; set; }
4      public string Name { get; set; }
5      public string Filepath { get; set; }
6  }
```

Listing 10: C# Entität - LogDestination

Listing 10 definiert die LogDestination-Entität. Diese Entität repräsentiert ein Ziel, an das Log-Nachrichten geschrieben werden können – beispielsweise eine Datei oder eine Datenbank. Die Eigenschaft Filepath gibt dabei den konkreten Speicherort an.

```
public class LoggingLibrary
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int DefaultLogLevelId { get; set; }

    public LogLevel DefaultLogLevel { get; set; } = null!;

    public ICollection<LogLevel> LogLevels { get; set; } = new List<LogLevel>();
}
```

Listing 11: C# Entität - LoggingLibrary

Listing 11 zeigt die LoggingLibrary-Entität, die Informationen über die verwendeten Logging-Bibliotheken (z.B. Serilog, log4net) enthält. Neben den grundlegenden Eigenschaften wie Id und Name werden hier auch Beziehungen zu anderen Entitäten modelliert. Die Eigenschaft DefaultLogLevelId und die dazugehörige Navigationsproperty DefaultLogLevel ermöglichen den direkten Zugriff auf das standardmäßig konfigurierte LogLevel dieser Bibliothek. Zudem wird die Sammlung LogLevels initialisiert, um alle von der Logging-Bibliothek unterstützten LogLevel aufzunehmen. Navigationsproperties wie DefaultLogLevel und LogLevels erleichtern den Zugriff auf zusammenhängende Daten, ohne dass zusätzliche Datenbankabfragen erforderlich sind.

```
public class LogLevel
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

Listing 12: C# Entität - LogLevel

Listing 12 präsentiert die LogLevel-Entität, welche die unterschiedlichen Schweregrade von Log-Nachrichten abbildet. Die Eigenschaft Id dient als eindeutiger Identifikator, während Name den Namen des LogLevels (wie Debug, Info oder Error) definiert. Diese Entität wird sowohl einzeln verwendet als auch als Teil von Beziehungen in anderen Entitäten, wie beispielsweise in der LoggingLibrary-Entität.

```
public class Site
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Filepath { get; set; }
    public int FiletypeId { get; set; }
    public int CurrentLoggingLibraryId { get; set; }
    public int CurrentLogLevelId { get; set; }

    public Filetype Filetype { get; set; } = null!;
    public LoggingLibrary CurrentLoggingLibrary { get; set; } = null!;
    public LogLevel CurrentLogLevel { get; set; } = null!;

    public ICollection<LogDestination> LogDestinations { get; set; } = new
    ↪ List<LogDestination>();
}
```

Listing 13: C# Entität - Site

Listing 13 definiert die zentrale Site-Entität, die eine von der Programmierfabrik verwaltete Webseite repräsentiert. Neben grundlegenden Eigenschaften wie Id, Name und Filepath werden hier auch Beziehungen zu anderen Entitäten abgebildet. Die Eigenschaften FiletypeId, CurrentLoggingLibraryId und CurrentLogLevelId verknüpfen die Site mit dem entsprechenden Dateityp, der aktuell verwendeten Logging-Bibliothek und dem aktuell gesetzten LogLevel. Die Navigationsproperties Filetype, CurrentLoggingLibrary und CurrentLogLevel ermöglichen es, direkt auf die zugehörigen Objekte zuzugreifen, ohne dass erst eine separate Abfrage notwendig wäre. Zudem wird über die Sammlung LogDestinations modelliert, dass eine Site mehrere Ziele für Log-Nachrichten besitzen kann.

```
public static class ErrorMessages
{
    public const string SiteNotFound = "Site not found";
    public const string LogLevelNotFound = "Log level not found";
    public const string UnsupportedLogLevel = "The current logging library
    ↪ does not support the specified log level";
    public const string LogLevelUnchanged = "The specified log level is
    ↪ already set";
}
```

Listing 14: C# Konstantenklasse - ErrorMessages

Listing 14 definiert die statische Klasse ErrorMessages, welche eine zentrale Sammlung von konstanten Fehlermeldungen enthält. Diese Meldungen repräsentieren verschiedene domänenspezifische Fehlerfälle, beispielsweise wenn eine Seite nicht gefunden wird (Site not found), ein Log-Level

nicht vorhanden ist (”Log level not found”), das angeforderte Log-Level von der aktuell verwendeten Logging-Bibliothek nicht unterstützt wird (”The current logging library does not support the specified log level”) oder wenn das Log-Level bereits gesetzt ist (”The specified log level is already set”).

Diese Konstanten werden in der Infrastrukturschicht in Verbindung mit FluentResults verwendet, um bei auftretenden Fehlern konsistente und verständliche Fehlermeldungen zurückzugeben. Wird zum Beispiel in einem Service ein Fehler festgestellt, so wird mittels Result.Fail ein entsprechendes Fehlerobjekt erzeugt, dem eine der in ErrorMessages definierten Meldungen zugeordnet wird. Dadurch wird gewährleistet, dass Fehler im gesamten System einheitlich kommuniziert werden, was die Fehlerdiagnose und -behebung vereinfacht. Dieser Ansatz unterstützt auch das Prinzip des Domain-Driven Designs, indem er sicherstellt, dass die Geschäftslogik und zugehörige Fehlermeldungen zentral und konsistent definiert sind.

5.1.2 Anwendungsschicht

```
1 public record UpdateSiteLogLevelDto(int NewLogLevelId, int? ResetTimeMinutes =
  ↪  null);
```

Listing 15: C# Record - UpdateSiteLogLevelDto

Listing 15 zeigt das UpdateSiteLogLevelDto. Dieses Data Transfer Object (DTO) wird verwendet, wenn das Log-Level einer Seite geändert werden soll. Es enthält zwei Properties: NewLogLevelId, das die ID des neuen Log-Levels angibt, und ResetTimeMinutes, einen optionalen Parameter, der bestimmt, nach wie vielen Minuten das Log-Level wieder zurückgesetzt werden soll.

```
1 public class NotFoundError(string message) : Error(message);
2 public class InternalServerError(string message) : Error(message);
3 public class BadRequestError(string message) : Error(message);
```

Listing 16: C# Fluent-Fehlerklassen

Listing 16 definiert drei Fehlerklassen – NotFoundError, InternalServerError und BadRequestError – die von der FluentResults-Bibliothek genutzt werden, um standardisierte Fehlermeldungen zu erzeugen. Diese Fehler werden im Controller verwendet, um mittels der HandleError-Methode (siehe Listing 40) ein passendes ProblemDetails-Objekt zurückzugeben, wenn etwas schief läuft.

```
public interface ISiteService
{
    Task<Result<IEnumerable<SiteGetDto>>> GetAllSitesAsync();
    Task<Result<IEnumerable<LogLevel>>> GetSiteLogLevelsAsync(Guid siteId);
    Task<Result> UpdateSiteLogLevelAsync(Guid siteId, UpdateSiteLogLevelDto
    ↪ updateSiteLogLevelDto);
}
```

Listing 17: C# Interface - ISiteService

Listing 17 definiert das Interface ISiteService. Dieses Interface wird in der Infrastrukturschicht implementiert und beschreibt im Grunde, wie bestimmte Geschäftslogikoperationen umgesetzt werden sollen. Konkret legt es drei Methoden fest: GetAllSitesAsync zum Abrufen aller Seiten, GetSiteLogLevelsAsync zum Ermitteln der unterstützten Log-Level für eine bestimmte Seite sowie UpdateSiteLogLevelAsync zum Aktualisieren des Log-Levels einer Seite anhand eines UpdateSiteLogLevelDto. Durch die Verwendung dieses Interfaces wird sichergestellt, dass die Geschäftslogik klar definiert und kapselt ist. Unterschiedliche Implementierungen können so problemlos ausgetauscht werden, ohne dass sich die Schnittstellen zwischen den Schichten verändern.

```
public static class DomainToDtoMapper
{
    public static SiteGetDto ToGetDto(this Site site)
    {
        return new SiteGetDto(site.Id, site.Name);
    }
}
```

Listing 18: C# Mappingklasse - DomainToDtoMapper

Das Mapping von Domain-Objekten zu DTOs erfolgt in Listing 18. Die Erweiterungsmethode ToGetDto() transformiert ein Site-Objekt in ein SiteGetDto, sodass nur die wesentlichen Informationen – die eindeutige Kennung (Id) und der Name – an den Client übermittelt werden.

```
public record UpdateLogLevelStream(Guid SiteUid, string LogLevelName, string
↪ TargetName);
```

Listing 19: C# Record - UpdateLogLevelStream

Schließlich definiert Listing 19 das Record UpdateLogLevelStream. Dieses Kommando dient dazu, eine Änderung des Log-Levels einer Seite zu signalisieren. Es enthält drei Properties: SiteUid (die eindeutige Kennung der Seite), LogLevelName (den Namen des neuen Log-Levels)

und TargetName. Letzteres entspricht der LogDestination, also dem Ziel, wohin die Log-Daten letztlich gesendet oder gespeichert werden sollen. Die LogDestination ist eine Entität, die bereits im Abschnitt 4.2 ausführlich erklärt wurde.

5.1.3 Infrastrukturschicht

Application Database Context (AppDbContext)

Der AppDbContext stellt den zentralen Zugriff auf die Persistenzebene dar. Über ihn werden die Entitäten Site, LogLevel etc.) in der Datenbank verwaltet und Abfragen oder Änderungen durchgeführt. Intern nutzt er die Implementierung DatabaseHelper.

```
1  public async Task<Result<IEnumerable<SiteGetDto>>> GetAllSitesAsync()
2  {
3      return await DatabaseOperationHelper.HandleDatabaseOperationAsync(async ()
       ↪  =>
4      {
5          var sites = await dbContext.Sites
6              .AsNoTracking()
7              .Select(s => s.ToGetDto())
8              .ToListAsync();
9
10         return Result.Ok(sites.AsEnumerable());
11     });
12 }
```

Listing 20: C# Methodenimplementierung - GetAllSitesAsync (Service)

Listing 20 zeigt eine Methode, die alle Seiten als IEnumerable<SiteGetDto> zurückliefert. Auffällig ist hierbei, dass der gesamte Datenbankzugriff in einen Aufruf der Methode HandleDatabaseOperationAsync eingebettet ist. Diese Methode übernimmt die Ausführung des übergebenen Funktionsaufrufs, der im Erfolgsfall ein Result.Ok mit den abgerufenen Daten zurückgibt.

```
public static async Task<Result<T>>
    HandleDatabaseOperationAsync<T>(Func<Task<Result<T>>> operation)
{
    try
    {
        return await operation();
    }
    catch (Exception ex)
    {
        return Result.Fail<T>(new InternalServerError(ex.Message));
    }
}
```

Listing 21: C# Methodenimplementierung - HandleDatabaseOperationAsync

Die Implementierung der HandleDatabaseOperationAsync-Methode (siehe Listing 21) zeigt die generische Variante, die in Fällen verwendet wird, in denen ein Rückgabewert erwartet wird. Im Fehlerfall wird innerhalb eines try-catch-Blocks die Exception abgefangen und in ein Result.Fail<T> umgewandelt, wobei die Exception Message als Fehlermeldung übermittelt wird. Dies ermöglicht dem Client, genau zu erkennen, was schief gelaufen ist, ohne dass die Exception direkt weitergereicht wird.

Da nicht alle Operationen einen Rückgabewert liefern, existiert zusätzlich eine nicht-generische Variante der HandleDatabaseOperationAsync-Methode, die lediglich ein Result zurückgibt. In der vorliegenden Methode (Listing 20) wird jedoch die generische Variante verwendet, da ein IEnumerable<SiteGetDto> erwartet wird. Diese Unterscheidung zwischen generischen und nicht-generischen Varianten sorgt für eine einheitliche und robuste Fehlerbehandlung im gesamten System.

```
public static class RedisStreamConfig
{
    public const string LogChangeUpdateKey = "LogChangeUpdate";
    public const string LogChangeUpdateStream =
        "LogActivatorLogChangeStream";
}
```

Listing 22: C# Konstantenklasse - RedisStreamConfig

Listing 22 definiert die Konfiguration für den Redis-Stream als eine statische Klasse namens RedisStreamConfig. In dieser Klasse werden zwei Konstanten definiert: LogChangeUpdateKey mit dem Wert “LogChangeUpdate” und LogChangeUpdateStream mit dem Wert “LogActivatorLogChangeStream”. Diese Konstanten geben den Schlüssel und das zugehörige Feld an, die für das Schreiben von Log-Änderungsbefehlen in den Redis-Stream verwendet werden.

```
private async Task WriteStreamIntoRedis(UpdateLogLevelStream
    ↪ updateLogLevelStream)
{
    var db = redis.GetDatabase();

    await db.StreamAddAsync(RedisKeys.LogChangeUpdateKey,
        RedisKeys.LogChangeUpdateStream,
        ↪ JsonConvert.SerializeObject(updateLogLevelStream));
}
```

Listing 23: C# Methodenimplementierung - WriteStreamIntoRedis

Listing 23 zeigt, wie dieser Command in den Redis-Cache geschrieben wird. Ein Redis-Stream ist eine spezielle Datenstruktur, die zum Speichern von zeitlich geordneten Ereignissen verwendet wird. In einem Redis-Stream werden neue Einträge stets am Ende angehängt, sodass die chronologische Reihenfolge der Ereignisse erhalten bleibt. In diesem Beispiel wird zuerst über redis.GetDatabase() eine Verbindung zur Redis-Datenbank aufgebaut. Anschließend wird die Methode StreamAddAsync aufgerufen, die drei Argumente erwartet: Das erste Argument ist der RedisKey, also der Schlüssel, unter dem der Stream gespeichert wird (in diesem Fall RedisKeys.LogChangeUpdateKey). Das zweite Argument ist ein RedisValue, der das Feld im Stream angibt (hier RedisKeys.LogChangeUpdateStream), und das dritte Argument ist ebenfalls ein RedisValue, der den eigentlichen Wert enthält – in diesem Fall wird der UpdateLogLevelStream als JSON-String serialisiert. Nachdem der Command erfolgreich in den Redis-Stream geschrieben wurde, kann ein externer Service diesen erkennen und auswerten, um die entsprechende Änderung im Log-Level durchzuführen.

```
127.0.0.1:6379> keys *
1) "LogChangeUpdate"
```

Listing 24: Redis - Auflisten aller Schlüssel

Listing 24 zeigt ein Beispiel aus der Redis-Konsole, in dem der Schlüssel “”LogChangeUpdate“’ aufgeführt wird. Dieser Schlüssel entspricht dem in der Infrastruktur definierten RedisStreamConfig.LogChangeUpdateKey (siehe Listing 22). Mit dem Befehl keys * werden alle in der Redis-Datenbank vorhandenen Schlüssel aufgelistet. Das Vorhandensein des Schlüssels “LogChangeUpdate” bestätigt, dass der zuvor über die Methode StreamAddAsync (siehe Listing 23) in den Redis-Stream geschriebene Command erfolgreich im Cache gespeichert wurde.

```
127.0.0.1:6379> type "LogChangeUpdate"
stream
```

Listing 25: Redis – Typ des LogChangeUpdate-Schlüssels

Listing 25 zeigt, dass der Schlüssel “LogChangeUpdate” vom Typ stream ist. Dieser Befehl, der in der Redis-Konsole ausgeführt wurde, gibt als Antwort den Typ des Schlüssels zurück und bestätigt damit, dass die zuvor in den Redis-Cache geschriebene Nachricht in Form eines Redis-Streams gespeichert wurde. Die Tatsache, dass Redis den Schlüssel als stream identifiziert, unterstreicht die Verwendung der speziellen Datenstruktur, die für das zeitlich geordnete Speichern von Ereignissen konzipiert ist. Dies ist besonders nützlich für Szenarien wie Logging oder die asynchrone Übermittlung von Befehlen an externe Services, da so sichergestellt wird, dass die Reihenfolge der Ereignisse erhalten bleibt.

```
1  127.0.0.1:6379> xread streams "LogChangeUpdate" 1
2  1) 1) "LogChangeUpdate"
3     2) 1) 1) "1738959959291-0"
4           2) 1) "LogActivatorLogChangeStream"
5              2) "{\"SiteUid\":\"c5183c9d-6cb9-466b-8760-6ea43efd22a4\",
6              \"LogLevelName\":\"Info\",\"TargetName\":\"System Logs\"}"
7        2) 1) "1738959959294-0"
8           2) 1) "LogActivatorLogChangeStream"
9              2) "{\"SiteUid\":\"c5183c9d-6cb9-466b-8760-6ea43efd22a4\",
10             \"LogLevelName\":\"Info\",\"TargetName\":\"Error Logs\"}"
```

Listing 26: Redis - Wert des LogChangeUpdate-Schlüssels

Listing 26 zeigt die Ausgabe des Redis-Befehls xread für den Stream mit dem Schlüssel “LogChangeUpdate”. Mit diesem Befehl wird der Inhalt des Streams ausgelesen, wobei in diesem Fall zwei Einträge zurückgegeben werden.

In der Ausgabe sieht man, dass zunächst der Name des Streams, also “LogChangeUpdate”, ausgegeben wird (Zeile 2). Anschließend folgt eine Liste von Einträgen. Der erste Eintrag, identifiziert durch die ID “1738959959291-0” (Zeile 3), enthält als Feld den Namen “LogActivatorLogChangeStream” (Zeile 4) und als zugehörigen Wert einen JSON-String (Zeilen 5–6). Dieser JSON-String repräsentiert ein UpdateLogLevelStream-Objekt, welches die Eigenschaften SiteUid, LogLevelName und TargetName beinhaltet. Dabei gibt TargetName an, wohin die Log-Daten letztlich gesendet werden sollen. Im vorliegenden Beispiel ist TargetName auf “System Logs” gesetzt.

Der zweite Eintrag, identifiziert durch die ID “1738959959294-0” (Zeile 7), weist ebenfalls das Feld “LogActivatorLogChangeStream” (Zeile 8) auf, enthält jedoch einen JSON-String, in dem TargetName auf “Error Logs” gesetzt ist (Zeile 10).

```
1  public async Task<Result> UpdateSiteLogLevelAsync(Guid siteId,
   ↪  UpdateSiteLogLevelDto updateSiteLogLevelDto)
2  {
3      var siteResult = await
       ↪  GetSiteWithLoggingLibraryAndLogDestinationsAsync(siteId);
4      if (siteResult.IsFailed)
5          return siteResult.ToResult();
6
7      var site = siteResult.Value;
8      var previousLogLevelId = site.CurrentLogLevelId;
9
10     if (previousLogLevelId == updateSiteLogLevelDto.NewLogLevelId)
11         return Result.Fail(new
           ↪  BadRequestError(ErrorMessages.LogLevelUnchanged));
12
13     if (!IsSupportedLogLevel(site, updateSiteLogLevelDto.NewLogLevelId))
14         return Result.Fail(new
           ↪  BadRequestError(ErrorMessages.UnsupportedLogLevel));
15
16     site.CurrentLogLevelId = updateSiteLogLevelDto.NewLogLevelId;
17
18      var updateResult = await ApplyLogLevelChangeAsync(site,
        ↪  updateSiteLogLevelDto.NewLogLevelId);
19     if (updateResult.IsSuccess
20         && updateSiteLogLevelDto.ResetTimeMinutes is > 0)
21     {
22         ScheduleLogLevelReset(siteId, previousLogLevelId,
           ↪  updateSiteLogLevelDto.ResetTimeMinutes.Value);
23     }
24     return updateResult;
25 }
```

Listing 27: C# Methodenimplementierung - UpdateSiteLogLevelAsync (Service)

Im Folgenden wird die zentrale Logik zur Änderung des Log-Levels einer Seite vorgestellt. Die Implementierung zeigt, wie die Geschäftsregeln und Fehlerprüfungen in der Infrastrukturschicht umgesetzt wurden und wie externe Dienste, wie ein Job-Queue-Mechanismus (Hangfire), zur asynchronen Verarbeitung genutzt werden.

Listing 27 zeigt die Methode UpdateSiteLogLevelAsync. Diese Methode dient dazu, ein neues Log-Level für eine Seite zu setzen. Zunächst wird über die Methode GetSiteWithLoggingLibraryAndLogDestinationsAsync die entsprechende Seite samt zugehöriger Logging-Bibliothek und Log-Destinations abgerufen. Scheitert dieser Abruf, wird ein Fehler-Result zurückgegeben. Anschließend wird geprüft, ob das neue Log-Level bereits dem aktuell gesetzten Log-Level entspricht (Zeile 10). Falls dies der Fall ist, wird ein Fehler (Bad Request) zurückgegeben (Zeile 11), da keine Änderung erfolgt. Zudem wird mit der Methode IsSupportedLogLevel (siehe Listing

30) überprüft, ob das angeforderte Log-Level von der aktuell verwendeten Logging-Bibliothek unterstützt wird (Zeile 13). Ist beides erfüllt, wird das Log-Level in der Entität aktualisiert und über ApplyLogLevelChangeAsync (siehe Listing 31) in der Datenbank persistiert. Wird der Update-Vorgang erfolgreich abgeschlossen und liegt ein Reset-Intervall (in Minuten) vor, wird zusätzlich ein verzögerter Job über Hangfire geplant, um das Log-Level nach Ablauf des Zeitraums automatisch zurückzusetzen. Dieser Job wird in der Methode ScheduleLogLevelReset (siehe Listing 28) eingerichtet.

```
1 private static void ScheduleLogLevelReset(Guid siteId, int previousLogLevelId,
  ↪ int resetTimeMinutes)
2 {
3     BackgroundJob.Schedule<SiteService>(
4          service => service.ResetSiteLogLevelAsync(siteId,
           ↪ previousLogLevelId),
5         TimeSpan.FromMinutes(resetTimeMinutes));
6 }
```

Listing 28: C# Methodenimplementierung - ScheduleLogLevelReset

Für das automatische Zurücksetzen des Log-Levels wird Hangfire eingesetzt. Listing 28 zeigt die Methode ScheduleLogLevelReset. Hierbei wird ein verzögerter Job geplant, der nach Ablauf des angegebenen Zeitraums die Methode ResetSiteLogLevelAsync im SiteService aufruft. Hangfire übernimmt dabei die zuverlässige Ausführung des Jobs, sodass das Log-Level nach der definierten Zeit automatisch wieder auf den vorherigen Wert zurückgesetzt wird.

```
1  public async Task<Result> ResetSiteLogLevelAsync(Guid siteId, int
   ↪ resetLogLevelId)
2  {
3      var siteResult = await
       ↪ GetSiteWithLoggingLibraryAndLogDestinationsAsync(siteId);
4      if (siteResult.IsFailed)
5          return siteResult.ToResult();
6
7      var site = siteResult.Value;
8      if (site.CurrentLogLevelId == resetLogLevelId)
9          return Result.Ok();
10
11     site.CurrentLogLevelId = resetLogLevelId;
12     return await ApplyLogLevelChangeAsync(site, resetLogLevelId);
13 }
```

Listing 29: C# Methodenimplementierung - ResetSiteLogLevelAsync

Listing 29 zeigt die Methode ResetSiteLogLevelAsync, die vom verzögerten Hangfire-Job aufgerufen wird. Diese Methode setzt das Log-Level einer Seite zurück, sofern es vom aktuellen Wert abweicht, und wendet die Änderung über die Methode ApplyLogLevelChangeAsync an.

```
1  private static bool IsSupportedLogLevel(Site site, int logLevelId)
2  {
3      return site.CurrentLoggingLibrary.LogLevels.Any(ll => ll.Id ==
       ↪  logLevelId);
4  }
```

Listing 30: C# Methodenimplementierung - IsSupportedLogLevel

Die Methode IsSupportedLogLevel (siehe Listing 30) prüft, ob das angeforderte Log-Level von der aktuellen Logging-Bibliothek der Seite unterstützt wird. Dies geschieht, indem in der Liste der Log-Level der Logging-Bibliothek nach einem Eintrag mit der entsprechenden ID gesucht wird.

```
1  private async Task<Result> ApplyLogLevelChangeAsync(Site site, int logLevelId)
2  {
3      return await DatabaseOperationHelper.HandleDatabaseOperationAsync(async ()
       ↪  =>
4      {
5          var logLevel = await dbContext.LogLevel.FindAsync(logLevelId);
6          if (logLevel is null)
7              return Result.Fail(new
               ↪  BadRequestError(ErrorMessages.LogLevelNotFound));
8
9          await UpdateLogDestinationsAsync(site, logLevel);
10         await dbContext.SaveChangesAsync();
11         return Result.Ok();
12     });
13 }
```

Listing 31: C# Methodenimplementierung - ApplyLogLevelChangeAsync

Die Methode ApplyLogLevelChangeAsync (siehe Listing 31) führt die eigentliche Aktualisierung des Log-Levels in der Datenbank durch. Zuerst wird das neue Log-Level in der Datenbank gesucht. Falls das Log-Level nicht gefunden wird, wird ein Fehler zurückgegeben. Anschließend wird die Methode UpdateLogDestinationsAsync aufgerufen, die über alle Log-Destinations der Seite iteriert und für jede einen Command erstellt, der in den Redis-Stream geschrieben wird. Nachdem die Änderungen an den Log-Destinations vorgenommen wurden, wird die Datenbank gespeichert.

```
1 private async Task UpdateLogDestinationsAsync(Site site, LogLevel logLevel)
2 {
3     foreach (var logDestination in site.LogDestinations)
4     {
5         var updateLogLevelStream = new UpdateLogLevelStream(site.Id,
          ↪  logLevel.Name, logDestination.Name);
6         await WriteStreamIntoRedis(updateLogLevelStream);
7     }
8 }
```

Listing 32: C# Methodenimplementierung - UpdateLogDestinationsAsync

Schließlich zeigt Listing 32 die Methode UpdateLogDestinationsAsync. Hier wird für jede LogDestination der Seite ein UpdateLogLevelStream-Command erstellt, der über eine Hilfsmethode in den Redis-Stream geschrieben wird. Der UpdateLogLevelStream-Command enthält neben der Seiten-ID und dem neuen Log-Level auch den Namen der LogDestination. Dieser Mechanismus ermöglicht es einem externen Service, die Änderungen zu erkennen und entsprechend umzusetzen.

```
1 builder.Services.AddHangfire(c =>
2 {
3     c.UseSqlServerStorage(builder.Configuration
4     .GetConnectionString("Hangfire"));
5 });
6 builder.Services.AddHangfireServer();
```

Listing 33: C# - Hangfire hinzufügen

Listing 33 zeigt, wie Hangfire in der Anwendung integriert wurde. In Zeile 3 wird festgelegt, dass Hangfire seine Daten in einer SQL Server-Datenbank speichert, um sicherzustellen, dass alle Hintergrundjobs auch nach einem Serverabsturz oder Neustart erhalten bleiben. Der Aufruf von builder.Services.AddHangfireServer() in Zeile 5 registriert einen Hangfire-Server, der die gespeicherten Jobs asynchron abarbeitet.

```
1 app.UseHangfireDashboard();
```

Listing 34: C# - Hinzufügen des Hangfire Dashboards

Listing 34 zeigt, wie das Hangfire-Dashboard in die Anwendung integriert wurde. Mit dem Befehl app.UseHangfireDashboard() wird das Dashboard aktiviert, sodass es über die URL http://localhost:5082/hangfire aufgerufen werden kann. Über dieses Dashboard können alle geplanten und ausgeführten Hintergrundjobs visuell überwacht werden.

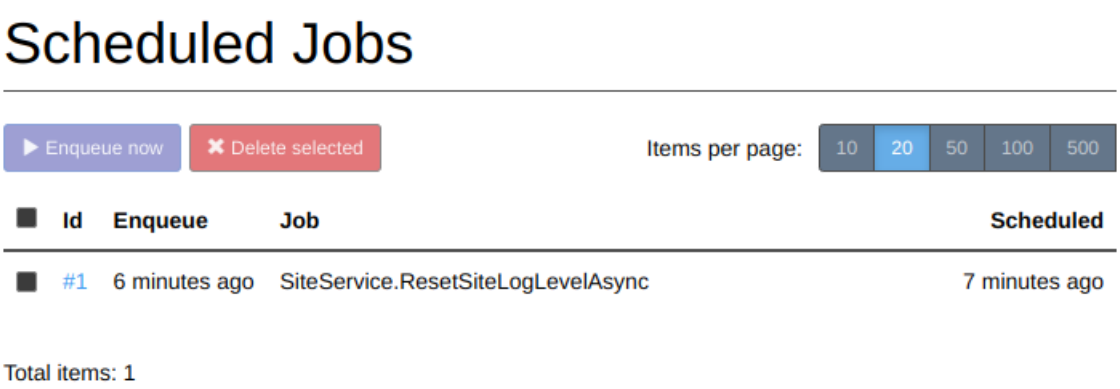


Abbildung 34: Beispiel - Hangfire Scheduled Jobs

Abbildung 34 zeigt einen Screenshot des Dashboards, in dem ein Scheduled Job zu sehen ist. Dieser Scheduled Job wird hinzugefügt, wenn die Methode ScheduleLogLevelReset (siehe Listing 28) aufgerufen wird, um das Log-Level einer Seite nach Ablauf einer definierten Zeit automatisch zurückzusetzen. Das Hinzufügen eines solchen Jobs im Dashboard unterstreicht die Zuverlässigkeit und Transparenz der asynchronen Hintergrundverarbeitung in der Anwendung.

5.1.4 API-Schicht

Im folgenden Abschnitt wird die Implementierung der API-Schicht detailliert erläutert. Dabei wird insbesondere auf drei Endpunkte eingegangen, die im Rahmen des Anwendungsfalls zum Ändern des LogLevels einer Seite realisiert wurden. Die Implementierung folgt dabei dem Prinzip, dass alle Operationen asynchron ausgeführt werden. Dies ist vor allem deshalb wichtig, weil asynchrone Methoden (gekennzeichnet durch async und den Rückgabewert Task<IActionResult>) erlauben, auf I/O-Operationen wie Datenbankabfragen zu warten, ohne den ausführenden Thread zu blockieren. Dadurch wird die Skalierbarkeit der Anwendung verbessert und es kann eine höhere Anzahl von gleichzeitigen Anfragen effizient bearbeitet werden.

```
1 [HttpGet]
2 [ProducesResponseType(StatusCodes.Status200OK, Type =
  ↪ typeof(IEnumerable<SiteGetDto>))]
3 public async Task<IActionResult> GetAllSitesAsync()
4 {
5     var result = await siteService.GetAllSitesAsync();
6     return result.IsSuccess
7         ? Ok(result.Value)
8         : HandleError(result.Errors.First(),
          ↪ StatusCodes.Status500InternalServerError);
9 }
```

Listing 35: C# Methodenimplementierung - GetAllSitesAsync (Controller)

Der erste Endpunkt GetAllSitesAsync (Listing 35) dient dazu, alle Seiten (Sites) zurückzuliefern, die von der Programmierfabrik verwaltet werden. Dabei wird ein Data Transfer Object (DTO) verwendet, welches alle relevanten Informationen, konkret die eindeutige Identifikation (UUID) und den Namen der Seite, enthält. Dieses DTO wird im Frontend genutzt, um dem Administrator eine Übersicht aller Seiten anzuzeigen. Diese Übersicht ist essenziell, da der Administrator vor der Änderung des LogLevels einer Seite zunächst die konkrete Seite auswählen muss. Sollte bei der Abfrage der Daten aus der Datenbank ein Fehler auftreten – beispielsweise durch einen Verbindungsabbruch – wird mittels der HandleError-Methode ein Fehlerobjekt verarbeitet und ein HTTP-Statuscode 500 (Internal Server Error) zurückgegeben. Somit wird dem Client in standardisierter Form ein Problem zurückgemeldet (Siehe Listing 41).

```
[
  {
    "id": "fcba4884-8d5b-4037-b5ca-54d5f1605c09",
    "name": "webconnector"
  },
  {
    "id": "c5183c9d-6cb9-466b-8760-6ea43efd22a4",
    "name": "gw.stammportal.at"
  },
  {
    "id": "6c671dbd-745d-4b9c-b6a6-e158f514b586",
    "name": "proxies"
  }
]
```

Listing 36: Beispielhafte JSON-Antwort - GetAllSitesAsync

Listing 36 zeigt den JSON-Response, der zurückgegeben wird, wenn die Methode erfolgreich ausgeführt wurde. Der Response enthält eine Liste von Objekten, die jeweils eine Seite (Site) repräsentieren. Jede Seite wird durch zwei Eigenschaften beschrieben: id und name. Die id ist eine eindeutige Identifikation (UUID), mit der die Seite eindeutig referenziert werden kann. Der name gibt den Namen der Seite an, wie sie in der Anwendung bekannt ist. Im gezeigten Beispiel handelt es sich um Testdaten, die zur Veranschaulichung der API-Antwort dienen. In einer echten Anwendung würden hier die tatsächlich gespeicherten Seiten der Programmierfabrik zurückgegeben werden. Diese Daten sind besonders für das Frontend wichtig, da sie genutzt werden, um eine Auswahl an verfügbaren Seiten anzuzeigen. Der Administrator kann dann eine dieser Seiten auswählen, um beispielsweise deren Log-Level zu ändern oder weitere Informationen abzurufen.

```
[HttpGet("{siteId:guid}/logLevels")]
[ProducesResponseType(StatusCodes.Status200OK, Type =
↪  typeof(IEnumerable<LogLevel>))]
[ProducesResponseType(typeof(ProblemDetails), StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetSiteLogLevelsAsync(Guid siteId)
{
    var result = await siteService.GetSiteLogLevelsAsync(siteId);
    return result.IsSuccess
        ? Ok(result.Value)
        : HandleError(result.Errors.First());
}
```

Listing 37: C# Methodenimplementierung - GetSiteLogLevelsAsync (Controller)

Der zweite Endpunkt GetSiteLogLevelsAsync (Listing 37) hat den Zweck, für eine ausgewählte Seite alle validen LogLevels zurückzuliefern. Da sich die gültigen LogLevels abhängig von der verwendeten Logging-Library unterscheiden, erfolgt hier eine dynamische Bestimmung der möglichen LogLevels. Dies ist insbesondere wichtig, weil ein Administrator, der das LogLevel einer Seite ändern möchte, zunächst wissen muss, welche LogLevels überhaupt unterstützt werden. Sollte für die angegebene Seiten-ID (GUID) keine Seite gefunden werden, wird ein HTTP-Statuscode 404 (Not Found) zurückgegeben. Auch hier wird im Fehlerfall das standardisierte Problem (wie in der HandleError-Methode beschrieben) verwendet. Kommt es hingegen zu einem schwerwiegenden Fehler bei der Datenbankabfrage, so wird ebenfalls der Fehlercode 500 zurückgegeben. Die Entscheidung, in diesem Fall die Entity direkt zurückzuliefern (anstelle eines DTO), beruht darauf, dass die Entity alle notwendigen Informationen enthält, die dem Client präsentiert werden sollen.

```json
[
  {
    "id": 1,
    "name": "Debug"
  },
  {
    "id": 2,
    "name": "Info"
  },
  {
    "id": 3,
    "name": "Error"
  },
  {
    "id": 4,
    "name": "Warn"
  },
  {
    "id": 6,
    "name": "Fatal"
  }
]
```

Listing 38: Beispielhafte JSON-Antwort - GetSiteLogLevelsAsync

Listing 38 zeigt einen Beispiel-Response für die Methode GetSiteLogLevelsAsync. Die Antwort besteht aus einer Liste von Objekten, die jeweils ein Log-Level repräsentieren. Jedes Objekt enthält eine id, die das Log-Level eindeutig identifiziert, und einen name, der die Bezeichnung des Log-Levels angibt. Es ist auffällig, dass die id 5 nicht vorhanden ist. Der Grund dafür liegt darin, dass jede Seite eine andere Logging-Bibliothek verwenden kann, die möglicherweise unterschiedliche Log-Level definiert. In diesem Beispiel existieren die Log-Level “Debug”, “Info”, “Error”, “Warn” und “Fatal”, jedoch fehlt das Log-Level mit der ID 5, da es in der verwendeten Konfiguration nicht existiert.

```
[HttpPut("{siteId:guid}/updateLogLevel")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(typeof(ProblemDetails), StatusCodes.Status404NotFound)]
[ProducesResponseType(typeof(ProblemDetails),
    StatusCodes.Status400BadRequest)]
public async Task<IActionResult> UpdateSiteLogLevelAsync(Guid siteId,
    [FromBody] UpdateSiteLogLevelDto updateSiteLogLevelDto)
{
    var result = await siteService.UpdateSiteLogLevelAsync(siteId,
        updateSiteLogLevelDto);
    return result.IsSuccess
        ? NoContent()
        : HandleError(result.Errors.First());
}
```

Listing 39: C# Methodenimplementierung - UpdateSiteLogLevelAsync (Controller)

Der dritte Endpunkt UpdateSiteLogLevelAsync (Listing 39) ermöglicht es, das aktuell konfigurierte LogLevel einer Seite zu ändern. Die Validierung des Vorgangs erfolgt in mehreren Schritten: Zunächst wird überprüft, ob die Seite mit der angegebenen GUID existiert. Falls dies nicht der Fall ist, wird ein 404-Fehler (Not Found) zurückgegeben. Weiterhin wird geprüft, ob das gewünschte neue LogLevel sinnvoll ist. So kommt es zu einem HTTP-Statuscode 400 (Bad Request), wenn beispielsweise das neue LogLevel bereits dem aktuell gesetzten LogLevel entspricht oder wenn das neue LogLevel von der derzeit verwendeten Logging-Library nicht unterstützt wird. Wie in den anderen Endpunkten, sorgt die HandleError-Methode für die einheitliche Fehlerbehandlung, sodass im Falle eines Problems stets ein standardisiertes Problem zurückgegeben wird. Sollte während der Aktualisierung ein Fehler im Datenbankzugriff (z.B. durch Entity Framework) auftreten, wird dies ebenfalls mit einem HTTP-Statuscode 500 signalisiert.

```
private ObjectResult HandleError(IError error, int? statusCode = null)
{
    var resolvedStatusCode = statusCode ?? error switch
    {
        NotFoundError => StatusCodes.Status404NotFound,
        BadRequestError => StatusCodes.Status400BadRequest,
        _ => StatusCodes.Status500InternalServerError
    };

    return Problem(
        detail: error.Message,
        statusCode: resolvedStatusCode,
        instance: HttpContext.Request.Path,
        title: ReasonPhrases.GetReasonPhrase(resolvedStatusCode)
    );
}
```

Listing 40: C# Methodenimplementierung - HandleError

Die Methode HandleError (Listing 40) spielt in der gesamten API-Schicht eine zentrale Rolle, da sie als Standardmechanismus zur Fehlerbehandlung implementiert wurde. Anstatt im Fehlerfall unterschiedliche Fehlerformate zu liefern, wird hier stets ein Problem zurückgegeben. Dieses Vorgehen orientiert sich an dem RFC-Standard (vgl. [5]), welcher das Problem Details for HTTP APIs beschreibt. Der Vorteil dieser standardisierten Fehlerdarstellung liegt in der Konsistenz und Nachvollziehbarkeit der Fehlermeldungen: Jeder Client, der diese API konsumiert, kann einheitlich auf Fehler reagieren und beispielsweise automatisch Fehlerlogs erstellen oder entsprechende Benutzerhinweise anzeigen. In der HandleError-Methode wird anhand des übergebenen Fehlerobjekts (welches unterschiedliche Fehlerklassen wie NotFoundError oder BadRequestError repräsentieren kann) ein entsprechender HTTP-Statuscode bestimmt. Sollte kein spezifischer Code übergeben werden, wird über ein switch-Statement der Statuscode festgelegt. Anschließend wird mittels der ProblemDetails-Methode ein standardisiertes Fehlerobjekt erzeugt, das neben dem Fehlertext auch den Pfad der Anfrage und den zugehörigen Statuscode enthält.

```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.6.1",
  "title": "Internal Server Error",
  "status": 500,
  "detail": "A network-related or instance-specific error occurred ...",
  "instance": "/api/sites",
  "traceId": "00-6023c916b2c502abab4f112f5fe5ee8b-2afbb2e7adeea3a5-00"
}
```

Listing 41: Beispielhafte JSON-Fehlerantwort gemäß RFC 7807

Listing 41 zeigt beispielhaft den Output, der durch die HandleError-Methode (siehe Listing 40) erzeugt wird. Der Output entspricht dem RFC-7807-Standard, der eine einheitliche Fehlerdarstellung in HTTP-APIs vorsieht. Der JSON-Output ist so strukturiert, dass er alle relevanten Informationen zum Fehler übersichtlich darstellt. Im Folgenden wird jeder Key im JSON-Objekt erläutert.

Type

In Zeile 1 wird der Key "type" definiert. Dieser enthält eine URI, die den Typ des Fehlers spezifiziert und auf weiterführende Informationen im entsprechenden RFC-Abschnitt verweist. Diese URI dient als Referenzpunkt, über den Clients zusätzliche Details zum Fehler abrufen können.

Title

Zeile 2 zeigt den Key "title", welcher eine kurze, prägnante Beschreibung des Fehlers liefert. Hier wird beispielsweise " Internal Server Error" ausgegeben, was dem Client sofort signalisiert, dass ein Serverfehler aufgetreten ist.

Status

In Zeile 3 wird der Key "status" definiert. Dieser gibt den HTTP-Statuscode an, der den Fehler charakterisiert – in diesem Fall 500, was für einen internen Serverfehler steht. Dieser Statuscode ermöglicht es Clients, programmgesteuert auf verschiedene Fehlersituationen zu reagieren.

Detail

Der Key "detail" in Zeile 4 liefert eine detaillierte Beschreibung des Fehlers. Anhand dieser Information, beispielsweise "A network-related or instance-specific error occurred ...", erhält der Client zusätzliche Kontextinformationen, die insbesondere bei der Fehlerdiagnose und -behebung hilfreich sind.

Instance

In Zeile 5 wird der Key "instance" definiert, welcher den konkreten Pfad der Anfrage angibt, die den Fehler verursacht hat. Dies hilft dabei, den Fehler direkt mit dem entsprechenden API-Endpunkt zu verknüpfen – in diesem Fall "/api/sites".

TraceId

Schließlich wird in Zeile 6 der Key “traceId” ausgegeben. Diese eindeutige Kennung erlaubt es, den Fehler in Log-Dateien oder über verteilte Tracing-Systeme zurückzuverfolgen, was die Fehlersuche und -analyse erheblich erleichtert.

5.1.5 Logging & Monitoring

In diesem Abschnitt wird gezeigt, wie Logging in dieser Anwendung umgesetzt wurde. Die hier vorgestellte Implementierung baut auf den theoretischen Grundlagen auf (siehe Abschnitt 2.1.2), in denen die Konzepte des strukturierten Loggings und der zentralen Fehlerüberwachung ausführlich erläutert wurden. Ziel ist es, ein Logging-System zu realisieren, das es ermöglicht, wichtige Ereignisse der Anwendung nicht nur in der Konsole darzustellen, sondern auch an ein zentrales Logmanagement-System zu senden, um eine bessere Überwachung und Analyse zu gewährleisten.

```
1  Log.Logger = new LoggerConfiguration()
2      .Enrich.FromLogContext()
3      .WriteTo.Console()
4      .WriteTo.OpenTelemetry(x =>
5      {
6          x.Endpoint = builder.Configuration["Seq:Endpoint"];
7          x.Protocol = OtlpProtocol.HttpProtobuf;
8          x.Headers = new Dictionary<string, string>
9          {
10             ["X-Seq-ApiKey"] = builder.Configuration["Seq:ApiKey"]!,
11         };
12         x.ResourceAttributes = new Dictionary<string, object>
13         {
14             ["service.name"] = builder.Configuration["Seq:ServiceName"]!
15         };
16     })
17     .CreateLogger();
18
19 builder.Services.AddSerilog();
```

Listing 42: C# – Backend-Logger-Konfiguration

Der folgende Codeausschnitt (Listing 42) zeigt, wie Serilog[50] in die Anwendung integriert wurde. Mithilfe von Serilog werden Logeinträge mit zusätzlichen Kontextinformationen angereichert, was die spätere Fehlerdiagnose und Analyse erleichtert. Über die Methode WriteTo.Console() werden Logeinträge in der Konsole angezeigt, was insbesondere während der Entwicklungsphase

[50] https://docs.datalust.co/docs/using-serilog

von großem Nutzen ist. Gleichzeitig sorgt die Konfiguration über WriteTo.OpenTelemetry() dafür, dass die Logs im OpenTelemetry-Format an einen OTLP-Endpunkt gesendet werden. In dieser Implementierung erfolgt die Übertragung an Seq – ein zentrales Logmanagement-System, das die zentrale Speicherung und Analyse der gesammelten Logs ermöglicht. [51]

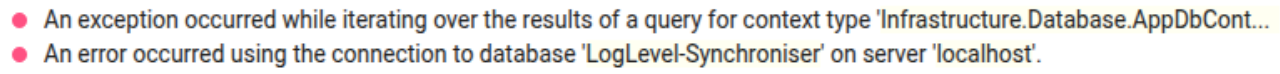


Abbildung 35: Beispiel – Fehlerlogs in Seq

Abbildung 35 zeigt die Fehlerlogs in Seq. In diesem Fall wurde ein Request an den Server gesendet, während gleichzeitig die Verbindung zur Datenbank getrennt war. Normalerweise würden alle Logeinträge angezeigt werden, doch aufgrund der Konfiguration, die die Anzeige auf Fehler beschränkt, werden hier nur die kritischen Fehlerereignisse dargestellt. Dies ermöglicht eine gezielte Analyse der Fehlerursache und vereinfacht die Fehlersuche im Produktionsumfeld.

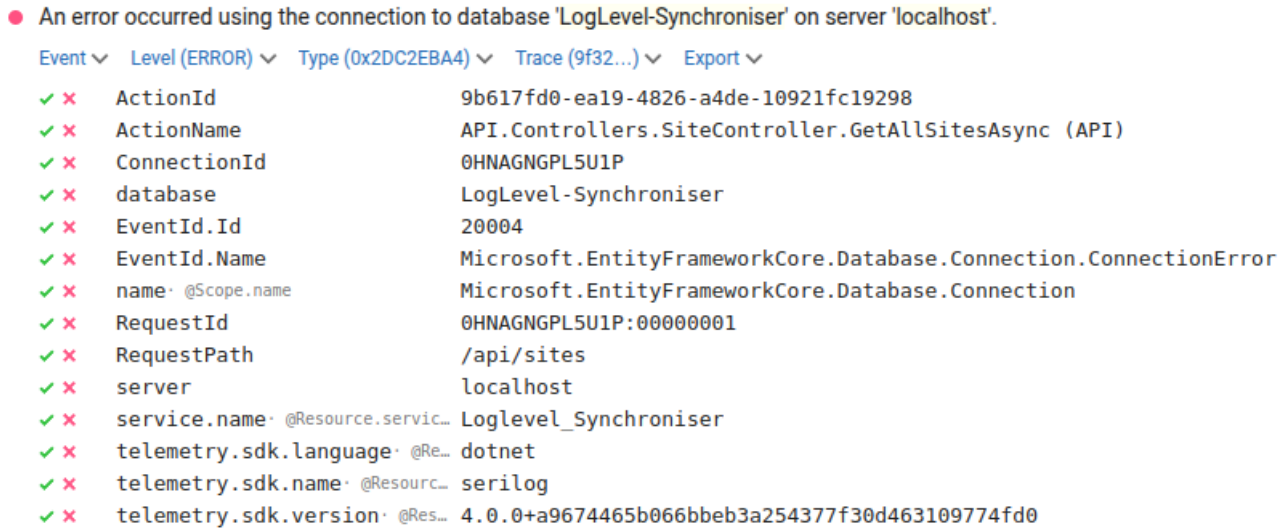


Abbildung 36: Beispiel - Informationen über einen Fehlerlog in Seq

Abbildung 36 zeigt die detaillierten Informationen zu einem spezifischen Fehler in Seq. Nachdem auf einen Fehler geklickt wurde, werden zusätzliche Metadaten sichtbar, die bei der Analyse und Nachverfolgung des Fehlers helfen. Es werden verschiedene Schlüssel angezeigt, die wichtige Informationen enthalten: So liefert der Schlüssel ActionId eine eindeutige Kennung der ausgeführten Aktion, während ActionName den Namen der entsprechenden API-Methode angibt. Die ConnectionId identifiziert die verwendete Datenbankverbindung, und der Schlüssel database nennt die betroffene Datenbank. Mit EventId.Name wird der Name des ausgelösten Ereignisses, beispielsweise ein Fehler bei der Datenbankverbindung, angegeben. Weitere Informationen

[51] https://datalust.co/seq

ergeben sich aus dem name, das den Namespace oder Bereich beschreibt, in dem der Fehler aufgetreten ist, und der RequestId, die eine eindeutige ID für die HTTP-Anfrage darstellt. Der RequestPath zeigt den aufgerufenen API-Endpunkt, der den Fehler verursacht hat, während server angibt, auf welchem Server die fehlerhafte Anfrage bearbeitet wurde. Zusätzlich wird der Name des Dienstes, der den Fehler protokolliert hat, unter service.name aufgeführt. Abschließend liefern die Schlüssel telemetry.sdk.language, telemetry.sdk.name und telemetry.sdk.version Informationen über die Programmiersprache, den Namen und die Version des Telemetrie-SDKs, das für die Erfassung der Logs genutzt wurde. Diese zusammenhängende Darstellung der Metadaten ermöglicht es, den Fehler im Kontext der gesamten Systemarchitektur zu analysieren und trägt so wesentlich zur schnellen Identifikation und Behebung von Problemen bei.

5.2 Frontend-Implementierung

5.2.1 Webframework

Als Webframework wird die Python Library Flask, siehe 2.4.2, verwendet. Flask ist leichtgewichtig und man kann damit leicht, mit wenig Source Code eine Webapplikation programmieren.

```
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def home():
7      dropdowns = [
8          {"id": "choose-sites", "label": "Site", "text": "Select site",
           ↪ "content_id": "sites"},
9          {"id": "choose-log-level", "label": "Log level", "text": "Select log
           ↪  level", "content_id": "log-levels"},
10         {"id": "chosen-sleep-time", "label": "Reset", "text": "Select
           ↪  minutes", "content_id": "minutes"}
11     ]
12     return render_template('index.html', dropdowns=dropdowns)
13
14 if __name__ == '__main__':
15     app.run(debug=True)
```

Listing 43: Python - Flask Grundaufbau

In Listing 43 sieht man den Grundaufbau der Flask Applikation. Nachdem von der zuvor via PIP, siehe 2.4.5, installierten Python Library flask, die Klasse Flask und die Funktion render_template importiert worden sind, wird ein Flask Objekt erstellt und als app gespeichert. Über app können

später alle Routen, Einstellungen und Ähnliches konfiguriert werden. Zuerst wird die eine Route, welche die Applikation hat, durch die Funktion home definiert. Die Funktion liefert das, in der Datei index.html geschriebene HTML zurück, nachdem die von Flask zur Verfügung gestellten Funktion render_template den Jinja Code mit dem Parameter dropdowns zu reinem HTML umgewandelt hat. Mithilfe eines Dekorators wird die Funktion app zugewiesen und den Pfad zu dieser Funktion definiert. Letztlich muss app noch gestartet werden, welches mittels der Methode run passiert.

5.2.2 Statische Dateien

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <link rel="stylesheet" href="/static/w3.css">
        <script src="/static/jquery-3.7.1.min.js"></script>
        <script src="/static/script.js"></script>
    </head>
    <body>
        <img src="/static/logo.svg">
    </body>
</html>
```

Listing 44: HTML - statische Dateien einbinden

Flask bietet standardmäßig den Endpunkt /static/{Pfad} zur Verfügung. Über dieser URL werden alle statische Dateien, welche index.html benötigt und sich im Ordner static befinden, bereitgestellt. Die Verwendung der Dateien sieht man in Listing 44

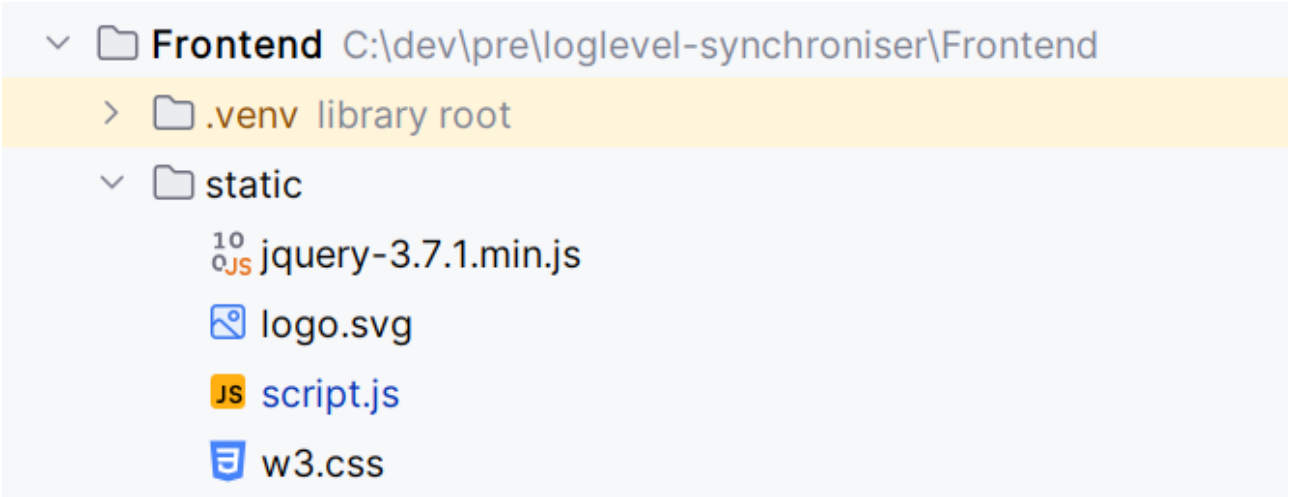


Abbildung 37: Statische Dateien

Wie man in Abbildung 37 sieht, werden einige Dateien zur Verfügung gestellt.

- jquery-3.7.1.min.js: gekürzte Datei für jQuery, siehe 2.4.6
- logo.svg: das Logo von Programmierfabrik GmbH
- script.js: jQuery Code, für die Backend Kommunikation
- w3.css: Datei für W3.CSS, siehe 2.4.4

5.2.3 Templates

Templates sind HTML Dokumente, in welche Jinja Code eingebaut ist und welche von flask mit render_template zu reinem HTML aufgelöst werden.

```
1  <!doctype html>
2  <html lang="en">
3  {% include 'head.html' %}
4  <body>
5
6  {% include 'navbar.html' %}
7
8  <div class="content">
9      {% block content %}{% endblock %}
10 </div>
11
12 </body>
13 </html>
```

Listing 45: base.html - HTML, Jinja

In Listing 45 ist die Datei „base.html“ abgebildet. Man sieht hier schon einige Anwendungen von Jinja. Auf normale Jinja Funktionen können mit „{% ... %}“ darauf zugegriffen werden, Variablen in Jinja kann man mit „{{ ... }}“ einfügen. Zuerst wird der Inhalt von „head.html“ an der richtigen Stellen eingefügt. Das funktioniert mittels include. Dann wird die Navbar eingefügt und anschließend ein Block mit dem Namen „content“ erstellt, welcher später ersetzt werden kann.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0,
          ↪  maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>{% block title %}Loglevel-Synchroniser{% endblock %}</title>
    <link rel="stylesheet" href="/static/w3.css">
    <script src="/static/jquery-3.7.1.min.js"></script>
    <script src="/static/script.js"></script>
</head>
```

Listing 46: head.html - HTML, Jinja

In Listing 46 sieht man den Inhalt von „head.html“. Diese Datei wird verwendet, um den head Bereich von dem restlichen Bereich abzugrenzen und wiederverwendbar zu machen.

```
<div class="w3-bar" style="background-color: #406e87; color: white">
    <div class="w3-bar-item">
        <img src="/static/logo.svg" alt="Programmierfabrik Hagenberg Logo">
    </div>
    <div class="w3-bar-item" style="padding: 0">
        <h1>Loglevel-Synchroniser</h1>
    </div>
</div>
```

Listing 47: navbar.html - HTML, Jinja

Auch der Code von der Navbar, welche überall in einer Applikation gleich bleiben soll, ist in eine eigene Datei „navbar.html“ ausgelagert.

```
{% extends 'base.html' %}

{% block content %}
<div class="w3-display-container" style="background-color: #818181; height:
↪ 48vh">
    <div class="w3-display-middle w3-display-container w3-card w3-white"
    ↪ style="height: 60%; width: 30%">
        <div class="w3-container w3-display-middle" style="width: 100%">

            {% include 'dropdown.html' %}

            ...

        </div>
    </div>
</div>
{% endblock %}
```

Listing 48: index.html - HTML, Jinja

In Listing 48 ist der Code von der Datei „index.html“ zu sehen. Es nimmt die Datei „base.html“ als eine Basis, mit „extends“ und erweitert diese, indem sie den Block „content“ mit Code füllt. Hierin wird dann wiederrum der Source Code von „dropdown.html“ eingefügt.

```
{% for dropdown in dropdowns %}
<div class="w3-row w3-padding">
    <div class="w3-col l4">
        <label for="{{ dropdown.id }}">{{ dropdown.label }}:</label>
    </div>
    <div class="w3-col l8">
        <div class="w3-dropdown-click" style="width: 100%">
            <button id="{{ dropdown.id }}" class="w3-button" type="button"
                    style="background-color: #406e87; color: white; width:
                    ↪ 100%">
                {{ dropdown.text }}
            </button>
            <div id="{{ dropdown.content_id }}" class="w3-dropdown-content
            ↪ w3-bar-block w3-card-4"></div>
        </div>
    </div>
</div>
{% endfor %}
```

Listing 49: dropdown.html - HTML, Jinja

In Listing 49 ist die Datei „dropdown.html“ abgebildet. Sie beginnt direkt mit einer for Schleife in Jinja, welche durch alle Dropdowns, welche im Python Code der Funktion render_template

übergeben worden sind, durchgeht und den HTML Code darin mehrmals untereinander dupliziert, mit einigen Anpassungen über die Variable „dropdown“, welche den die Informationen des momentanen Dropdowns speichert. Es wird Beispielsweise dem Label die Informationen des Dropdown übergeben.

5.2.4 Design

Zum designen des HTMLs wurde das CSS-Framework W3.CSS verwendet. Dieses stellt einige CSS Klassen zur Verfügung, welche es einem einfach machen ein ansprechendes und schlichtes Design zu erstellen.

```
<div
    class="w3-display-middle w3-display-container w3-card w3-white"
       style="height: 60%; width: 30%"
>
    ...
</div>
```

Listing 50: Card stylen mit W3.CSS

In Listing 50 sieht man wir die Card erstellt wird, auf welcher das Formular sich befindet. Mithilfe einiger CSS-Klassen, wie zum Beispiel “w3-card” oder “w3-white”, welche das div als Card und imt weißen Hintergrund kennzeichnen. Wie man sieht beginnen alle Klassen von W3.CSS mit dem Prefix “w3”.

Die Dropdown Menüs, welche sehr aufwändig mit normalem CSS zu erstellen wäre, wird ebenfalls mit W3.CSS einfach erstellt.

```
<div id="sites" class="w3-dropdown-content w3-bar-block w3-card-4"></div>
```

Listing 51: Dropdown Menü mit W3.CSS

Wie man in Listing 51 sieht, kann man mithilfe der CSS Klasse „w3-dropdown-content“ ein Dropdown erstellen. Befüllt mit Optionen wird es später mithilfe von jQuery und AJAX.

```
<div class="w3-bar" style="background-color: #406e87; color: white">
    <div class="w3-bar-item">
        <img src="/static/logo.svg" alt="Programmierfabrik Hagenberg Logo">
    </div>
    <div class="w3-bar-item" style="padding: 0">
        <h1>Loglevel-Synchroniser</h1>
    </div>
</div>
```

Listing 52: Header

In Listing 52 ist zu sehen, wie ein Header mit W3.CSS erstellt wird. In diesem Header ist zum einem das Logo der Programmierfabrik GmbH, zum anderen der Titel der Applikation, also „Loglevel-Synchroniser“. Das funktioniert, indem ein Container mit der Klasse „w3-bar“ existiert, in welcher alle einzelnen Items sind, welche nebeneinander im Header angezeigt werden sollen und mit der Klasse „w3-bar-item“ gekennzeichnet sind.

```
<div class="w3-row w3-padding">
    <div class="w3-col l4">
        <label for="choose-sites">Site:</label>
    </div>
    <div class="w3-col l8">
        <div class="w3-dropdown-click" style="width: 100%">
            <button id="choose-sites" class="w3-button" type="button"
                style="background-color: #406e87; color: white; width:
                ↪  100%">Select site
            </button>
        </div>
    </div>
</div>
```

Listing 53: Buttons in W3.CSS

Die Erstellung und Gestalltung der Buttons, welche die Dropdown Menüs öffnen, sind in Listing 53. Mit „w3-row“ werden die Child-Elemente nebeneinander und mit „w3-col“ untereinander angeordnet. Die Klassen „l4“ und „l8“ kennzeichnen die relative Größe innerhalb eines Containers.

```
<div class="w3-col l1">
    <input id="choose-sleep-time" class="w3-check" type="checkbox">
</div>
```

Listing 54: Checkbox in W3.CSS

Zur Erstellung einer Checkbox, die verwendet wird, um die optionale Auswahl einer Zeit zum Zurücksetzen der Log-Level Änderung freizuschalten, wird die W3.CSS Klasse „w3-check“ verwendet, so wie in Listing 54 abgebildet.

5.2.5 Backend Kommunikation und jQuery

Die Kommunikation zum Backend, um zum Beispiel die Liste der Seiten, welche verfügbar sind, zu bekommen, wird über AJAX mithilfe von jQuery verwendet. Der gesamte jQuery Quellcode befindet sich innerhalb von script.js, welches im „head“ Bereich von index.html eingebundet wird, so wie in Listing 44 zu sehen.

Innerhalb von jQuery gibt es die Funktion „$“, welche dazu verwendet wird, um auf die meisten Funktionalitäten, die jQuery zur Verfügung stellt, zu verwenden.

```
$(() => {
    fetchSites()
    fetchMinutes()
    ...
})
```

Listing 55: jQuery document ready Funktion

Damit viele jQuery Funktionen funktionieren, muss das Document Object Model (DOM) fertig geladen sein, sodass darauf zugegriffen werden kann. Übergibt man der $ Funktion, eine weitere Funktion, dann wird die übergebene Funktion erst aufgerufen, sobald alles fertig geladen ist, so wie in Listing 55 zu sehen. Hier werden dann vorab alle Informationen aus dem Backend geladen, welche nicht von etwas anderem abhängen, wie zum Beispiel die später ausgewählte Seite. Es werden also die verfügbaren Seiten, sowie die verschiedenen Minuten vom Backend angefordert.

```
function fetchSites() {
    const url = "http://localhost:5082/api/sites";
    $.ajax({
        url: url,
        success: (result, status, xhr) => {
            for (const site of result) {
                $("#sites").append(createDropdownOption(site.name, site.id));
            }
            addSiteDropdownClickListener();
        },
        error: (result, status, xhr) => {
            console.log(result);
            console.log(status);
            console.log(xhr);
        }
    });
}

```

Listing 56: fetchSites Funktion - jQuery

Wie in Listing 56 zu sehen, kann mittels $.ajax eine AJAX Abfrage an das Backend gesendet werden. Es wird die URL zu dem benötigtem Endpunkt und zwei Funktionen, welche steuern, was bei Erfolg beziehungsweise Misserfolg der Anfrage passieren soll, übergeben. Bei erfolgreichem Aufruf, wird durch die geladenen Seiten durch gegangen und alle Seiten dem Dropdown

hinzugefügt. Mittels „$(#sites)“ bekommt man das DOM Objekt mit der id „sites“, also das Dropdown und mit append kann man diesem dann ein DOM Objekt als Children anhängen. Das selbe gilt für die Funktion fetchMinutes.

```
function createDropdownOption(text, id) {
    const dropdownOption = $("<div>")
    dropdownOption.addClass("w3-bar-item w3-button")
    dropdownOption.text(text)
    dropdownOption.data("id", id)
    return dropdownOption
}
```

Listing 57: createDropdownOption Funktion - jQuery

Das Erstellen des DOM Objekts, welches eine Auswahl im Dropdown ist, passiert in der Funktion createDropdownOption, so wie in Listing 57 zu sehen. Mit $ kann man neue Objekte erstellen, indem das gewünschte Objekt in spitzen Klammern gesetzt und $ übergeben wird. Mit addClass wird diesem Objekt dann die gewollten CSS Klassen übergeben. Dann wird noch der Text, der Dropdown Option gesetzt und die id für später gespeichert.

```
function addSiteDropdownClickListener() {
    $("#sites>div").on("click", function () {
        const site = $(this)
        $("#choose-sites").text(site.text())
        chosenSite = site.data("id")
        chosenLogLevel = null
        $("#log-levels").empty()
        $("#choose-log-level").text("Select log level")
        $("#sites").toggleClass("w3-show")
        fetchLogLevels()
        $("#choose-log-level").removeAttr("disabled")
        $("#change-log-level").attr("disabled", "disabled")
    })
}
```

Listing 58: addSiteDropdownClickListener Funktion - jQuery

Jeder dieser Dropdown-Optionen benötigt zusätzlich eine Funktion, die ausgeführt wird, sobald sie angeklickt und damit ausgewählt wird. Im Listing 58 wird dies dargestellt. `$("#sites>div")` liefert alle `<div>`-Elemente innerhalb der ID `sites` zurück – also sämtliche Dropdown-Optionen. Mit der Methode `on` und dem ersten Argument `click` wird ein Click-Listener erstellt und zugewiesen. Im Unterschied zu fast sämtlichem anderem jQuery-Code wird hierbei die anonyme Funktion nicht mithilfe der Arrow-Function-Syntax `()=>{}`, sondern standardmäßig mit `function(){}` realisiert. Dies hat zur Folge, dass mittels `$(this)` auf das jeweils operierende

Objekt zugegriffen werden kann. Anschließend werden Texte von Elementen angepasst, manche Elemente deaktiviert und einzelne Attribute modifiziert, um einen reibungslosen Nutzererlebnisablauf zu gewährleisten. Zudem wird die Funktion `fetchLogLevels` aufgerufen, welche die Log-Levels lädt.

```
function fetchLogLevels() {
    const url = "http://localhost:5082/api/sites/" + chosenSite +
    ↪ "/logLevels";
    $.ajax({
        url: url,
        success: (result, status, xhr) => {
            for (const logLevel of result) {
                $("#log-levels").append(createDropdownOption(logLevel.name,
                ↪ logLevel.id));
            }
            addLogLevelDropdownClickListener();
        },
        error: (result, status, xhr) => {
            console.log(result);
            console.log(status);
            console.log(xhr);
        }
    });
}

```

Listing 59: fetchLogLevels Funktion - jQuery

Das Fetchen der Loglevel ist in Listing 59 zu sehen. Der Ablauf ist sehr ähnlich wie schon bei fetchSites, nur eben mit einigen Anpassungen. Es wird ein AJAX Aufruf gemacht, durch die Loglevel durch gegangen und Dropdown Optionen erstellt und der Click Listener erstellt.

```
function addLogLevelDropdownClickListener() {
    $("#log-levels>div").on("click", function () {
        const logLevel = $(this)
        $("#choose-log-level").text(logLevel.text())
        chosenLogLevel = logLevel.data("id")
        $("#log-levels").toggleClass("w3-show")
        let enabled = !$("#chosen-sleep-time").attr("disabled") === "disabled"
        if ((enabled && sleepTime != 0) || (!enabled)) {
            $("#change-log-level").removeAttr("disabled")
        }
    })
}
```

Listing 60: addLogLevelDropdownClickListener Funktion - jQuery

In Listing 60 ist die Funktion „addLogLevelDropdownClickListener“ abgebildet. Ähnlich wie addSiteDropdownClickListener kümmert sich diese Funktion um die Erstellung der Click Listener.

```
$(() => {
    ...
    $("#choose-sites").on("click", () => {
        $("#sites").toggleClass("w3-show")
    })
    $("#choose-log-level").on("click", () => {
        $("#log-levels").toggleClass("w3-show")
    })
    $("#change-log-level").on("click", () => {
        postChange()
    })
    $("#chosen-sleep-time").on("click", () => {
        $("#minutes").toggleClass("w3-show")
    })
    $("#choose-sleep-time").on("change", () => {
        toggleChooseSleepTime()
    })
})
```

Listing 61: Event Listener - jQuery

So wie in Listing 61 zu sehen, werden in der document ready Funktion außerdem einige Event Listener erstellt. Darunter das Anzeigen der Dropdown Menüs nach dem Klick auf den entsprechenden Buttons, das Verschicken der Loglevel Änderung und einiges mehr.

```
function toggleChooseSleepTime() {
    if ($("#chosen-sleep-time").attr("disabled") === "disabled") {
        $("#chosen-sleep-time").removeAttr("disabled")
        $("#change-log-level").attr("disabled", "disabled")
    } else {
        $("#chosen-sleep-time").attr("disabled", "disabled")
        sleepTime = 0
        $("#chosen-sleep-time").text("Select minutes")
        if (chosenLogLevel == null) {
            $("#change-log-level").attr("disabled", "disabled")
        } else {
            $("#change-log-level").removeAttr("disabled")
        }
    }
}
```

Listing 62: toggleChooseSleepTime Funktion - jQuery

Das komplexe Anzeigen und Aktivieren von Dropdown Menüs und Buttons, welche nach dem an ankreuzen der Checkbox passieren muss, wird durch die Funktion toggleChooseSleepTime gesteuert, so wie in Listing 62 zu sehen.

5.3 Testing & Qualitätssicherung

Testing und Qualitätssicherung spielen eine zentrale Rolle in der Softwareentwicklung. Ziel dieser Maßnahmen ist es, sicherzustellen, dass die entwickelten Komponenten wie erwartet funktionieren und den Anforderungen entsprechen. Durch automatisierte Tests können Fehler frühzeitig erkannt und behoben werden, wodurch die Stabilität und Wartbarkeit der Anwendung verbessert wird. Qualitätssicherung umfasst dabei nicht nur das Testen einzelner Module (Unit-Tests), sondern auch das Überprüfen der gesamten Systemintegration sowie die Durchführung von manuellen Tests, um eine hohe Softwarequalität zu gewährleisten.

5.3.1 Backend

```
1  public class SiteControllerTests
2  {
3      private readonly Mock<ISiteService> _mockSiteService;
4      private readonly SiteController _controller;
5
6      public SiteControllerTests()
7      {
8          _mockSiteService = new Mock<ISiteService>();
9          _controller = new SiteController(_mockSiteService.Object);
10
11         SetupControllerContext();
12     }
13
14     private void SetupControllerContext()
15     {
16         _controller.ControllerContext = new ControllerContext
17         {
18             HttpContext = new DefaultHttpContext()
19             {
20                 Request = { Path = "/api/sites" }
21             }
22         };
23     }
24 }
```

Listing 63: C# – Einrichtung des Testkontexts für den SiteController

Wie in den theoretischen Grundlagen in Abschnitt 2.1.7 ausführlich erläutert wurde, erlaubt das Mocking, das Verhalten von Abhängigkeiten zu simulieren. Dies ermöglicht das isolierte Testen einzelner Komponenten, ohne dass deren tatsächliche Implementierungen ausgeführt werden müssen. Listing 63 zeigt ein Beispiel für einen Unit-Test im Backend, der mit dem Testframework xUnit und dem Mocking-Framework Moq umgesetzt wurde. In diesem Beispiel wird ein Test für den SiteController erstellt. Dabei wird ein Mock-Objekt der Schnittstelle ISiteService erzeugt, welches in den Controller injiziert wird. Um eine NullPointerException zu vermeiden – die auftreten würde, wenn auf HttpContext.Request.Path in der HandleError-Methode zugegriffen wird (Siehe Listing 40 Zeile 13) – wird im Konstruktor die Methode SetupControllerContext() aufgerufen. Diese Methode initialisiert den ControllerContext des Controllers, indem ein DefaultHttpContext mit einem gesetzten Request-Pfad ("/api/sites") erstellt wird. Dadurch wird sichergestellt, dass der HttpContext vorhanden ist, wenn der Controller auf ihn zugreift, und das isolierte Testen des Controllers gelingt, ohne dass die tatsächliche Logik des Services ausgeführt wird.

```
1  [Fact]
2  public async Task GetAllSitesAsync_ReturnsOk_WhenSitesExist()
3  {
4      // Arrange
5      var mockSites = new List<SiteGetDto>
6      {
7          new(Guid.NewGuid(), "Test Site 1"),
8          new(Guid.NewGuid(), "Test Site 2"),
9          new(Guid.NewGuid(), "Test Site 3"),
10     };
11
12     _mockSiteService.Setup(s => s.GetAllSitesAsync())
13         .ReturnsAsync(Result.Ok<IEnumerable<SiteGetDto>>(mockSites));
14
15     // Act
16     var result = await _controller.GetAllSitesAsync();
17
18     // Assert
19     result.Should().BeOfType<OkObjectResult>();
20     var okResult = result as OkObjectResult;
21     okResult!.Value.Should().BeEquivalentTo(mockSites);
22 }
```

Listing 64: C# Unit-Test - GetAllSitesAsync (Service gibt alle Sites zurück)

Listing 64 zeigt einen Unit-Test für die Methode GetAllSitesAsync im SiteController. In Zeile 5 bis 10 wird eine Liste von Testdaten, bestehend aus drei SiteGetDto-Objekten, definiert. Diese Testdaten simulieren das Szenario, in dem Sites existieren. In Zeile 12 und 13 wird

das Verhalten des Mock-Objekts für die Schnittstelle ISiteService festgelegt, sodass bei einem Aufruf von GetAllSitesAsync() ein erfolgreiches Ergebnis mit den vordefinierten Testdaten zurückgegeben wird. Obwohl man in einer echten Umgebung erwarten würde, dass der Service mit der Datenbank kommuniziert, erfolgt hier keine Datenbankabfrage. Stattdessen wird ein vordefinierter Satz von Daten verwendet, um das Verhalten des Controllers isoliert zu testen. Der Test stellt sicher, dass, wenn der Service erfolgreich eine Liste von Sites zurückliefert, der Controller einen OkObjectResult zurückgibt (Zeile 19) und dass der zurückgegebene Wert exakt den Testdaten entspricht (Zeile 21).

```
1  [Fact]
2  public async Task GetAllSitesAsync_ReturnsProblem_WhenServiceFails()
3  {
4      // Arrange
5      var error = new InternalServerError("Something went wrong");
6      _mockSiteService.Setup(s => s.GetAllSitesAsync())
7              .ReturnsAsync(Result.Fail<IEnumerable<SiteGetDto>>(error));
8
9      // Act
10     var result = await _controller.GetAllSitesAsync();
11
12     // Assert
13     result.Should().BeOfType<ObjectResult>();
14     var problemResult = result as ObjectResult;
15     problemResult!.StatusCode.Should().Be(500);
16 }
```

Listing 65: C# Unit-Test - GetAllSitesAsync (Service gibt einen Fehler zurück)

Listing 65 zeigt einen Unit-Test für den Fall, dass beim Aufruf der Methode GetAllSitesAsync ein Fehler auftritt. In diesem Test wird mithilfe von Moq das Verhalten des ISiteService simuliert, sodass eine Fehlermeldung (ein InternalServerError mit der Nachricht “Something went wrong”) zurückgegeben wird. Anschließend wird die Methode GetAllSitesAsync aufgerufen (Zeile 10), und es wird überprüft, ob das Ergebnis vom Typ ObjectResult ist. In Zeile 15 wird sichergestellt, dass der HTTP-Statuscode des Ergebnisses 500 beträgt, was anzeigt, dass ein interner Serverfehler aufgetreten ist.

```
[Fact]
public async Task
    ↪ UpdateSiteLogLevelAsync_ReturnsBadRequest_WhenLogLevelUnchanged()
{
    // Arrange
    var siteId = Guid.NewGuid();
    var dto = new UpdateSiteLogLevelDto(1);
    var error = new BadRequestError("Log level unchanged");
    _mockSiteService.Setup(s => s.UpdateSiteLogLevelAsync(siteId, dto))
        .ReturnsAsync(Result.Fail(error));

    // Act
    var result = await _controller.UpdateSiteLogLevelAsync(siteId, dto);

    // Assert
    result.Should().BeOfType<ObjectResult>();
    var problemResult = result as ObjectResult;
    problemResult!.StatusCode.Should().Be(400);
}
```

Listing 66: C# Unit-Test - UpdateSiteLogLevelAsync (Log-Level unverändert)

Listing 66 zeigt einen Unit-Test für die Methode UpdateSiteLogLevelAsync im SiteController für den Fall, dass versucht wird, das Log-Level zu ändern, aber der neue Wert identisch mit dem aktuell gesetzten ist. In diesem Szenario wird ein Fehlerobjekt vom Typ BadRequestError mit der Nachricht “Log level unchanged” erzeugt. Das Mock-Objekt der Schnittstelle ISiteService wird so konfiguriert, dass es beim Aufruf von UpdateSiteLogLevelAsync ein gescheitertes Ergebnis (Result.Fail) mit diesem Fehler zurückliefert.

Anschließend wird die Methode UpdateSiteLogLevelAsync des Controllers aufgerufen, und es wird überprüft, ob das Ergebnis vom Typ ObjectResult ist. Über die Assert-Anweisungen wird sichergestellt, dass der zurückgegebene HTTP-Statuscode dem Wert 400 (Bad Request) entspricht, was darauf hinweist, dass ein Fehler aufgetreten ist, weil das Log-Level nicht geändert wurde.

Es sei angemerkt, dass in dieser Arbeit nicht alle Testmethoden im Detail gezeigt werden, da viele Tests nach demselben Prinzip aufgebaut sind. Das Grundprinzip bleibt dabei immer gleich: Mithilfe von Moq wird das Verhalten von Abhängigkeiten simuliert, sodass der Controller isoliert getestet werden kann, ohne dass tatsächlich auf externe Ressourcen (wie Datenbanken) zugegriffen wird.

5.3.2 Frontend

Da das Frontend keine neue Logik in das System einbaut, sondern nur die im Backend implementierte Logik ansteuert, wurde im Frontend getestet ob die korrekte Ansteuerung der Backend Methoden so wie gewollt funktioniert. Außerdem wurde manuell getestet, ob es irgendwelche visuellen Bugs in der Weboberfläche gibt, wie zum Beispiel, dass der Text auf einer Komponente, zu lang ist und diese Komponente überschreitet. All diese Möglichkeiten, welche die Benutzererfahrung verschlechtern würde, wurden identifiziert und behoben.

6 Ergebnis

In diesem Kapitel werden die Ergebnisse des Projekts zusammengefasst und bewertet. Es wird dargestellt, inwieweit die gesteckten Ziele erreicht wurden und wie die verschiedenen Komponenten miteinander harmonieren. Die Implementierung des LogLevel-Synchronisers zeigt, dass die in den theoretischen Grundlagen und im Design vorgestellten Konzepte erfolgreich in einer modularen und robusten Anwendung umgesetzt werden konnten.

6.1 Backend

6.2 Frontend

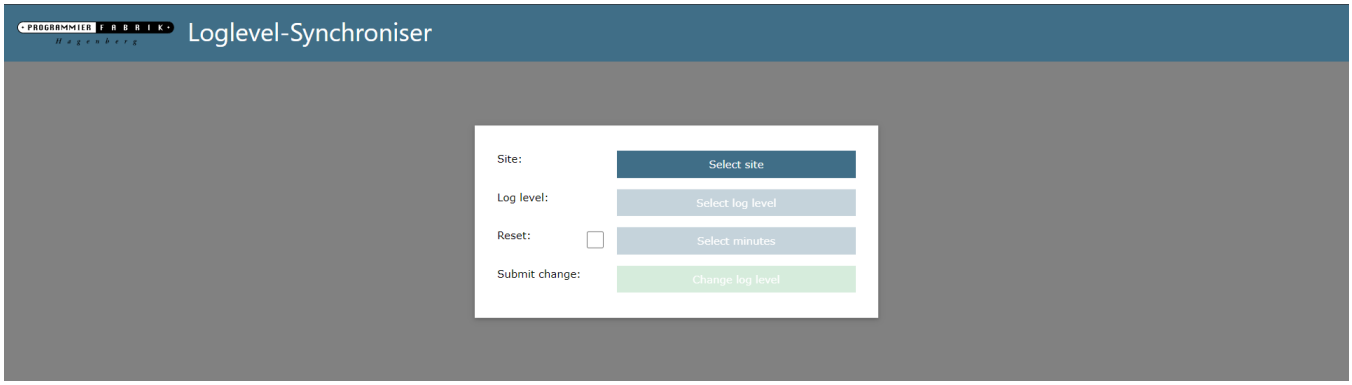


Abbildung 38: Loglevel-Synchroniser - Frontend

In Abbildung 38 ist das fertige Frontend abgebildet. Es wird so, wie von dem Betreuer der Diplomarbeit gefordert, ein schlichtes, einfaches und einheitliches Design, welches an vorherige Projekte der Programmierfabrik GmbH erinnert. In der Kopfzeile wird das Logo der Programmierfabrik angezeigt, sowie der Titel des Projekts. Die Hauptkomponente ist allerdings das Formular in der Mitte der Interfaces, worüber die gesamte, im Backend implementierte Funktionalität gesteuert werden kann. Man kann sich eine Seite aussuchen, das neue, gewünschte Log-Level dieser Seite festlegen, optional noch eine automatische Zurücksetzung aktivieren, mit einer personalisierten Wartezeit vor dem Zurücksetzen, und zum Schluss kann man die ausgewählte Log-Level Änderung bestätigen.

6.3 Ablauf

In diesem Abschnitt wird beschrieben, wie man die Anwendung verwendet.

Seite auswählen

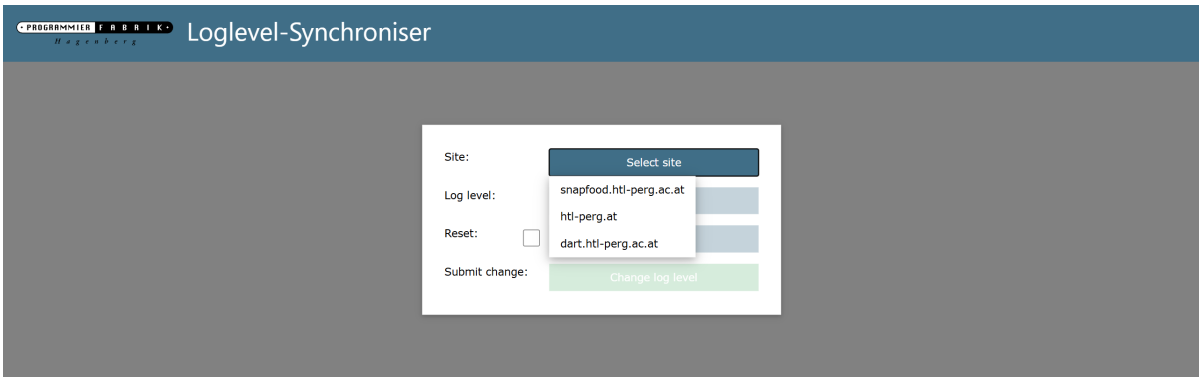


Abbildung 39: Seite auswählen

Der erste Schritt, um das Log-Level einer Seite zu ändern, ist die Seite auszuwählen, auf welcher die Änderung angewand werden soll. Das erfolgt durch das Klicken auf den Button „Select Site“. So wie man in 39 sieht, wird ein Dropdown Menü angezeigt, in welchem alle verschiedenen Seiten aufgelistet werden, welche am Server gespeichert sind. Die Abfrage nach den Seiten von dem Backend erfolgt mittels AJAX, siehe Abbschnitt 2.4.7, schon bevor man auf den Button überhaupt geklickt hat, sodass man nicht noch darauf warten, oder die Seite neu geladen werden muss. Das sorgt allerdings auch dafür, dass wenn man die Seite unabsichtlich neu lädt, die bereits ausgewählten Optionen wieder nicht ausgewählt werden. Nachdem man sich jetzt für eine Seite entschieden hat kann man zum Log-Level weitergehen.

Log-Level auswählen

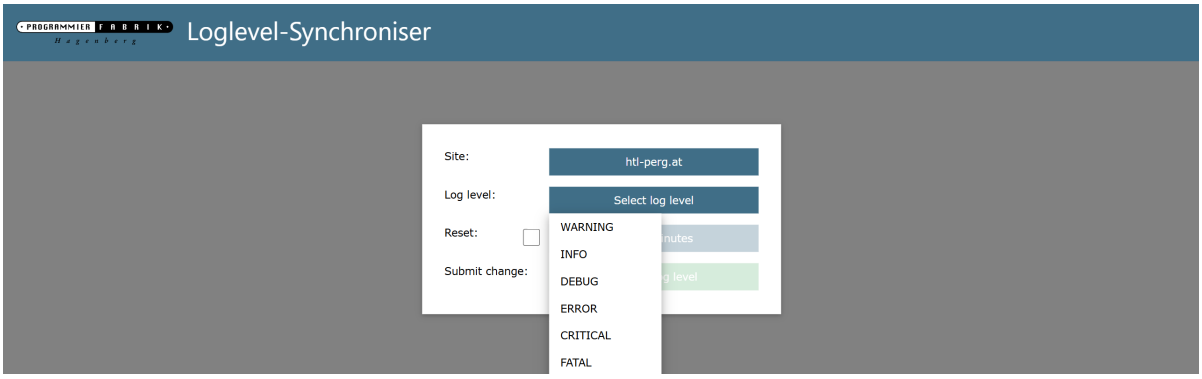


Abbildung 40: Log-Level auswählen

Durch das Auswählen der Seite wird zum einem der Text auf dem Button „Select Site“ zu der ausgewählten Seite geändert, wie man es in Abbildung 40 sieht, zum anderem wird so wie auch die Seiten geladen worden sind, die Loglevel dieser Seite mittels AJAX geladen. Das ist notwendig, weil verschiedene Seiten verschiedene Logging-Frameworks verwenden, welche über verschiedene Log-Level verfügen. Die Auswahl mittels des Dropdown Menüs erfolgt gleich wie beim Auswählen der Seite.

Optionale Reset Zeit auswählen

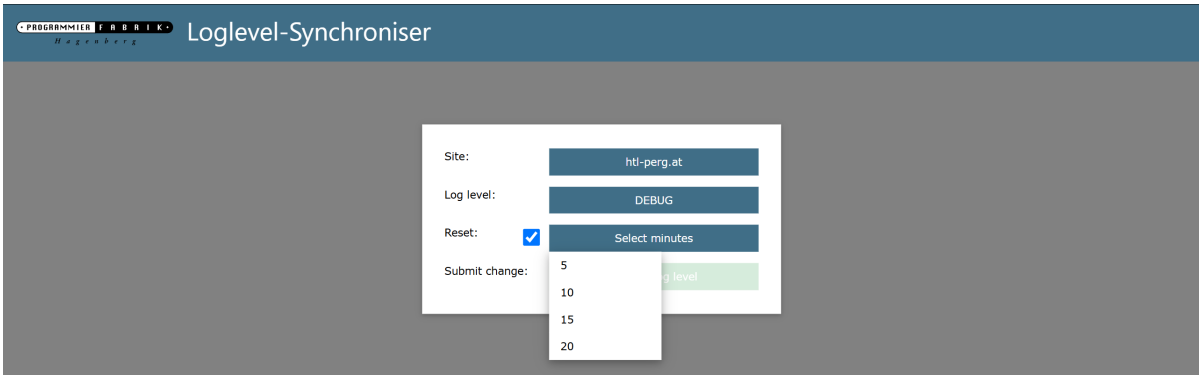


Abbildung 41: Reset Zeit auswählen

Man kann jetzt noch, wenn man die Checkbox aktiviert, optional eine Zeit auswählen, nach welcher, die ausgewählte Log-Level Änderung wieder zurück gesetzt wird. Das ist dahingehend nützlich, da viele Log-Level Änderungen nur für eine kurze Zeit passieren müssen, um zum Beispiel einen Fehler zu beheben.

Änderung speichern

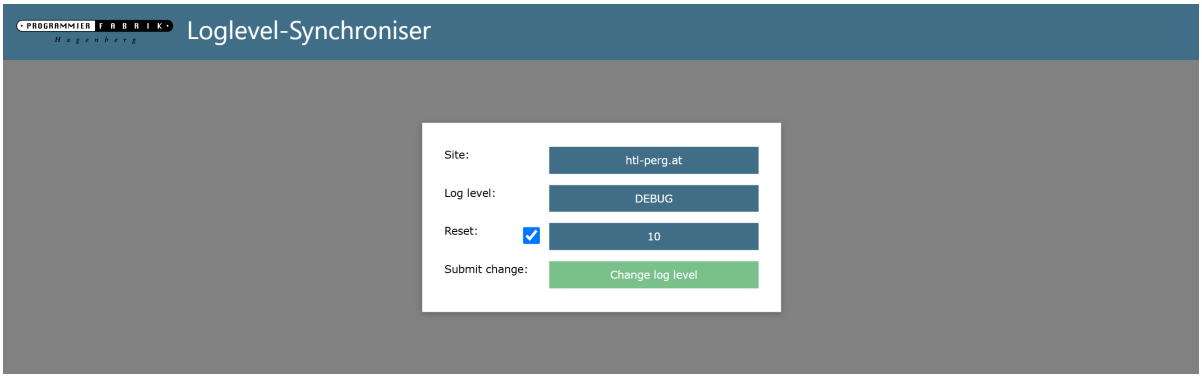


Abbildung 42: Änderung bestätigen

Zuletzt muss man jetzt nur noch die Änderungen, mithilfe des Buttons „Change log level“, bestätigen.

7 Resümees

In diesem Kapitel verfasst jedes der vier Teammitglieder ein eigenes Resümee. Hier werden individuelle Reflexionen über die Projektergebnisse, die Erfahrungen während der Umsetzung sowie die persönlichen Erkenntnisse zusammengefasst. Darüber hinaus wird eine Gesamtbewertung des Projekts vorgenommen, in der die zentralen Stärken und Schwächen, die Effektivität der eingesetzten Technologien und Methoden sowie mögliche Verbesserungsansätze diskutiert werden. Das Kapitel bietet somit einen umfassenden Überblick über die Lernerfahrungen, die erreichten Resultate und den Mehrwert, den das Projekt aus den unterschiedlichen Perspektiven der Teammitglieder bietet.

7.1 Jan-Niclas Hartl

Während meines vierwöchigen Praktikums bei der Programmierfabrik GmbH konnte ich, Jan-Niclas Hartl, wertvolle praktische Erfahrungen in der Softwareentwicklung und im agilen Projektmanagement sammeln. Wesentliche Elemente waren die täglichen, fünfminütigen Stand-up-Meetings und die wöchentlichen Retrospektiven, durch die ich die Vorteile agiler Methoden besser verstehen und unsere Arbeitsabläufe gezielt strukturieren konnte.

Ein wesentlicher Teil meiner Aufgaben bestand sowohl in der technischen Auseinandersetzung mit Redis als auch in der aktiven Zusammenarbeit mit der Gruppenleitung. Besonders im Bereich der Teamkoordination habe ich viel dazugelernt: Die Zusammenarbeit innerhalb der Gruppe gestaltete sich nicht immer reibungslos und erforderte eine klare Abstimmung aller Beteiligten. Um diese Hürden zu meistern, habe ich dazu beigetragen, den Kommunikationsfluss im Team zu verbessern und klare Strukturen zu schaffen. Durch die regelmäßigen Retrospektiven konnten zudem auftretende Schwierigkeiten reflektiert und passende Lösungen erarbeitet werden.

Insgesamt hat mir dieses Praktikum nicht nur neue technische Kenntnisse vermittelt, sondern mir auch wertvolle Einblicke in effektive Teamarbeit und Problemlösung gegeben. Durch neues Fachwissen und praxisnahe Arbeitsweisen konnte ich wertvolle Erkenntnisse für meine zukünftige berufliche Laufbahn gewinnen. Ich bin überzeugt, dass mir diese Erfahrungen helfen werden, in Zukunft strukturierter an Projekte heranzugehen und erfolgreich im Team zu agieren.

7.2 Maximilian Koch

In den vier Wochen meines Praktikums bei der Programmierfabrik GmbH sammelte ich, Maximilian Koch, wertvolle Erfahrungen in der Softwareentwicklung. Besonders spannend war für mich die Möglichkeit, an einem realen Projekt mitzuarbeiten, das später tatsächlich verwendet wird. Mein Schwerpunkt lag dabei auf der Integration der Datenbankanbindung im Backend.

Technisch stellte mich das Praktikum vor einige Herausforderungen, da viele Konzepte und Werkzeuge neu für mich waren. Zu Beginn war die Einarbeitung in die verschiedenen Technologien herausfordernd, doch mit der Zeit und durch den Austausch mit meinen Kollegen konnte ich mein Wissen stetig vertiefen.

Die Zusammenarbeit im Team verlief insgesamt gut, auch wenn es anfangs kleinere Anpassungsschwierigkeiten gab. Ich konnte mich jedoch schnell in die Abläufe einfinden und von den Erfahrungen meiner Kollegen profitieren. Besonders hilfreich war hierbei der regelmäßige Austausch.

Zusammenfassend hat mir das Praktikum viele wertvolle Erkenntnisse gebracht. Ich konnte praktische Erfahrungen in der Softwareentwicklung sammeln, und lernte, wie wichtig ein strukturiertes Vorgehen bei technischen Herausforderungen ist.

7.3 Eric Reps

Während meines vierwöchigen Praktikums bei der Programmierfabrik GmbH erhielt ich, Eric Reps einen umfassenden Einblick in die vielseitigen und zugleich anspruchsvollen Aufgaben der Softwareentwicklung. Mein Schwerpunkt lag auf der Implementierung zentraler Backend-Funktionen

Es fanden kurze tägliche Meetings und wöchentliche Retrospektiven zur Abstimmung statt. Im Vordergrund stand für mich die Programmierarbeit. Mit der Zeit wurde mir zunehmend bewusst, wie essenziell klare und effiziente Kommunikation im Team ist. Anfangs traten hierbei vereinzelt Schwierigkeiten auf, doch im Laufe des Praktikums konnten diese erfolgreich überwunden werden. Zudem unterstützte ich meine Kollegen gerne bei technischen Fragen. Dieser fachliche Austausch trug dazu bei, Herausforderungen gezielt anzugehen und zu bewältigen.

Rückblickend haben die während des Praktikums gesammelten Erfahrungen maßgeblich zur Weiterentwicklung meiner fachlichen als auch meiner teamorientierten Kompetenzen beigetragen. Nicht nur meine Programmierfähigkeiten wurden gestärkt, sondern ich erhielt auch wertvolle

Einblicke in die Arbeitsweise realer Softwareentwicklungsteams. Dabei konnte ich nachvollziehen, wie verschiedene Entwicklungsprozesse ineinandergreifen und wie man sich effektiv in ein Team integriert.

7.4 Luca Strobl

In meinem vierwöchigen Praktikum bei der Programmierfabrik GmbH konnte ich viel Berufserfahrung sammeln und einen umfassenden Einblick in die Abläufe der Projektorientierten Softwareentwicklung gewinnen.

Während der Zusammenarbeit habe ich festgestellt, dass die Einbindung in eine bestehende Teamstruktur eine Herausforderung ist. Verschiedene Arbeitsweisen und Kommunikationsstile haben anfangs zu Problemen geführt. Durch aktiven Austausch ist es mir jedoch gelungen, alles auf einen gemeinsamen Nenner zu bringen, sodass eine gute Zusammenarbeit funktioniert.

Während des Praktikums habe ich vor allem gelernt, mich an wechselnde Herausforderungen anzupassen. Besonders die Arbeit an komplexen Aufgabenstellungen hat mir gezeigt, wie wichtig eine strukturierte Herangehensweise ist, um effizient und zielgerichtet zu arbeiten. Hauptsächlich fokussierte ich mich auf die Implementierung des gesamten Frontends.

Rückblickend hat mir das Praktikum wertvolle Erkenntnisse in der Softwareentwicklung und in der Zusammenarbeit innerhalb eines Teams gegeben. Die gewonnenen Erfahrungen werden mir in zukünftigen Projekten helfen, mich schneller in neue Teams zu integrieren und effektiver an einem gemeinsamen Projekt zu arbeiten.

Abbildungsverzeichnis

Quellcodeverzeichnis

Glossar

API Eine Anwendungsprogrammierschnittstelle ist eine Sammlung von Methoden, Funktionen oder Routinen, die es unterschiedlichen Softwarekomponenten ermöglicht, miteinander zu kommunizieren.

ASP.NET Core Ein plattformübergreifendes, leistungsstarkes Framework zur Entwicklung moderner Webanwendungen und APIs.

Background Job Eine asynchrone Aufgabe, die im Hintergrund ausgeführt wird, um langlaufende oder zeitkritische Prozesse vom Hauptanwendungsfluss zu entkoppeln.

Configuration Die Gesamtheit der Einstellungen und Parameter, die das Verhalten einer Anwendung oder eines Systems steuern.

Database Ein strukturiertes Repository zur Speicherung von Daten, das in der Regel von einem Datenbankmanagementsystem (DBMS) wie SQL Server verwaltet wird.

DDD Domänengetriebenes Design ist ein Ansatz zur Softwareentwicklung, der den Fokus auf die Geschäftslogik und das fachliche Vokabular einer Anwendung legt. Dabei werden komplexe Probleme durch eine klare Trennung zwischen Domain, Anwendung und Infrastruktur gelöst.

DOM Das Document Object Model (DOM) ist eine standardisierte Schnittstelle, die HTML- oder XML-Dokumente als hierarchische Baumstruktur darstellt. Es ermöglicht die dynamische Manipulation und Aktualisierung von Webseiteninhalten.

DTO Ein Data Transfer Object ist ein einfaches Objekt, das ausschließlich dazu dient, Daten zwischen verschiedenen Schichten oder Systemen zu übertragen. Es enthält keine Geschäftslogik, sondern lediglich die benötigten Datenfelder.

Exception Ein Fehlerzustand, der während der Programmausführung auftritt und den normalen Ablauf unterbricht. Exceptions werden genutzt, um unerwartete oder fehlerhafte Zustände zu signalisieren.

Fluent API Ein Designansatz, der es ermöglicht, Konfigurationen und Abfragen durch eine verkettete, lesbare Methode zu definieren. Dies führt zu einer klaren und intuitiven Syntax.

FluentResults Eine Bibliothek für .NET, die eine flüssige API zur Verwaltung von Operationsergebnissen bietet. Sie kapselt Erfolge und Fehler in Result-Objekten, um eine konsistente Fehlerbehandlung zu ermöglichen.

GUID Eine weltweit eindeutige Nummer zur Identifikation von Objekten. Obwohl eine GUID häufig als Zeichenkette dargestellt wird, die sowohl Ziffern als auch Buchstaben enthält, liegt dies daran, dass sie im Hexadezimalsystem repräsentiert wird. Tatsächlich handelt es sich um eine sehr große Zahl, die so konstruiert ist, dass sie nahezu garantiert einzigartig ist.

Hangfire Eine Bibliothek zur Verarbeitung von Hintergrundaufgaben in .NET-Anwendungen. Hangfire ermöglicht das asynchrone und zuverlässige Ausführen von Jobs, die außerhalb des Hauptanwendungsflusses laufen.

Logging Der Prozess der Aufzeichnung von Ereignissen oder Zustandsänderungen innerhalb einer Anwendung. Logging dient der Überwachung, Fehlerdiagnose und Analyse des Anwendungsbetriebs.

Mapping Der Vorgang, bei dem Daten oder Objekte von einer Form in eine andere überführt werden, beispielsweise die Umwandlung von Domänenobjekten in Data Transfer Objects (DTOs) für die Kommunikation zwischen Schichten.

Moq Ein Mocking-Framework für .NET, das Entwicklern ermöglicht, Abhängigkeiten zu simulieren, um so Komponenten isoliert zu testen. Es hilft dabei, reale Implementierungen in Tests zu ersetzen.

Newtonsoft.Json Ein populäres JSON-Framework für .NET, auch bekannt als Json.NET, das zur Serialisierung und Deserialisierung von Objekten in das JSON-Format verwendet wird.

OTLP Das OpenTelemetry-Protokoll ist ein standardisiertes Protokoll zur Übertragung von Telemetriedaten (wie Traces, Metriken und Logs) zwischen instrumentierten Anwendungen und Observability-Backends.

ProblemDetails Ein standardisiertes Format für Fehlerantworten in HTTP APIs, wie es in RFC-7807 definiert ist. Es enthält Informationen wie Fehlercode, Titel, Detailbeschreibung und den betroffenen API-Pfad.

RAM Der Random Access Memory (RAM) ist der Arbeitsspeicher eines Computers, in dem Daten und Programme temporär gespeichert werden. Er ermöglicht einen schnellen Zugriff auf diese Daten während der Programmausführung.

Redis Ein leistungsstarker, in-Memory-Datenspeicher, der häufig als Cache oder Message Broker eingesetzt wird. Redis unterstützt verschiedene Datenstrukturen wie Listen, Mengen und Streams.

Redis Stream Eine Datenstruktur in Redis, die zur Speicherung einer geordneten Folge von Nachrichten verwendet wird. Sie eignet sich ideal für das Logging oder Messaging, da neue Einträge stets am Ende angehängt werden und die chronologische Reihenfolge erhalten bleibt.

RFC-7807 Ein Standard, der ein einheitliches Format für Fehlerbeschreibungen (ProblemDetails) in HTTP API-Antworten definiert.

Seq Ein zentrales Log-Management-System, das strukturierte Logs sammelt, speichert und visualisiert, um eine effiziente Fehlerdiagnose und Überwachung zu ermöglichen.

Serilog Eine strukturierte Logging-Bibliothek für .NET, die es ermöglicht, Logeinträge flexibel und konfigurierbar zu erfassen und zu verarbeiten.

xUnit Ein Framework für Unit-Tests in .NET, das zum Testen einzelner Codeeinheiten verwendet wird, um deren korrekte Funktionalität sicherzustellen.

Literaturverzeichnis

[1] A. Chuvakin, *Logging and Log Management: The Authoritative Guide to Dealing with Syslog, Audit Logs, Events, Alerts and Other IT'noise'.* Elsevier Science, 2012.

[2] D. Eddelbuettel, „A Brief Introduction to Redis," 2022. Online verfügbar: https://arxiv.org/abs/2203.06559

[3] S. Chacon und B. Straub, *Pro git.* Springer Nature, 2014.

[4] S. Dileepkumar und J. Mathew, „Optimize Continuous Integration and Continuous Deployment in Azure DevOps for a controlled Microsoft. NET environment using different techniques and practices," in *IOP Conference Series: Materials Science and Engineering*, Vol. 1085, Nr. 1. IOP Publishing, 2021, S. 012027.

[5] M. Nottingham und E. Wilde, „RFC 7807: Problem Details for HTTP APIs," 2016.