HAW HAMBURG

Bachelorarbeit

Peter Fischer

Audit Logging und Änderungsanalyse in datengetriebenen Webanwendungen

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Peter Fischer

Audit Logging und Änderungsanalyse in datengetriebenen Webanwendungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Henning Dierks
Zweitgutachter: Prof. Dr. Ulrike Herster

Eingereicht am: 18. September 2025

Peter Fischer

Thema der Arbeit

Audit Logging und Änderungsanalyse in datengetriebenen Webanwendungen

Stichworte

Audit Logging, Änderungsanalyse, Webanwendung, Benutzeraktionen, Nachvollziehbarkeit, ASP.NET Core, ASP.NET Boilerplate, Entity Framework, Softwarearchitektur

Kurzzusammenfassung

Diese Arbeit konzipiert ein Audit-Logging-System für eine datengetriebene Webanwendung. Das Ziel ist es, Änderungen und Benutzeraktionen zu erfassen und transparent darzustellen. Dazu wurde ein Modul entwickelt, das sowohl automatische als auch manuelle Protokollierung ermöglicht und strukturiert in einer Datenbank speichert. Die Ergebnisse zeigen, dass durch eine Kombination verschiedener Frameworks eine leicht erweiterbare Lösung zur Nachvollziehbarkeit von Benutzeraktionen realisiert werden kann.

Peter Fischer

Title of Thesis

Audit logging and change analysis in data-driven web applications

Keywords

Audit Logging, Change Analysis, Web Applications, User Actions, Traceability, ASP.NET Core, ASP.NET Boilerplate, Entity Framework, Software Architecture

Abstract

This bachelor thesis develops a concept for an audit logging system for a data-driven web application. The goal is to record changes and user actions and display them transparently. Therefore a module was developed that enables both automatic and manual logging and stores the data in a structured way in a database. The results show that a combination of different frameworks can be used to implement an expandable solution for tracking user actions.

Inhaltsverzeichnis

Listings **vii**

1 Einleitung **1**
1.1 Kundenprojekt . . . 2
1.2 Motivation und Ziele . . . 3

2 Grundlagen **4**
2.1 Begriffsdefinition . . . 4
2.2 Architektur von Webanwendungen . . . 4
2.2.1 Client & Server . . . 5
2.2.2 Frontend . . . 6
2.2.3 Backend . . . 10
2.2.4 Application Programming Interface . . . 11
2.3 Frameworks . . . 13
2.3.1 Angular . . . 13
2.3.2 .NET (Dotnet) . . . 14
2.3.3 Entity Framework . . . 16
2.3.4 ASP.NET Core . . . 16
2.4 Hypertext Transfer Protocol . . . 16
2.4.1 Status Codes . . . 17
2.4.2 Http-Kontext . . . 18
2.5 Relationale Datenbanken . . . 18
2.5.1 SQL-Datenbankoperationen . . . 19
2.6 Logging . . . 20
2.6.1 Audit-Log und Debug-Log . . . 20
2.6.2 Speicherformate . . . 21
2.6.3 Logging Kriterien . . . 21
2.6.4 Strukturierte Logs . . . 22
2.6.5 Unstrukturierte Logs . . . 23

2.6.6 Serilog 23

3 Logging-System **24**
3.1 Anforderungen 24
3.1.1 Separate Log-Datenbank 24
3.1.2 Logging-Modul 24
3.2 Konzept 28
3.2.1 Source Code Generation 29
3.2.2 Interceptors 29
3.2.3 Action-Filter 29
3.2.4 Event-Handler 30
3.3 Implementierung 30
3.3.1 Architektur 30
3.3.2 Hilfsklassen 35
3.4 Validierung 38
3.4.1 Manuelle Log-Einträge 38
3.4.2 Interceptor Log-Einträge 41
3.4.3 Exception Log-Einträge 42

4 API-Entwurf **44**
4.1 Welche Schnittstellen sind notwendig? 44
4.2 Implementierung einer Overview-API 45

5 Visualisierung **51**
5.1 Log-Übersicht 52
5.2 Log-Detailansicht 54

6 Log-Analyse **56**
6.1 Outlier 57
6.2 Automatische Analysemöglichkeiten 57
6.3 IQR (Interquartilsabstand) 58
6.4 Alternative Ausreißererkennung 60

7 Ergebnis **61**
7.1 Auditing-System 61
7.2 Bekannte Schwachstelle 63

8 Ausblick **64**

9 Fazit **65**

Literaturverzeichnis **66**

A Anhang **70**

A.1 Listings . . . 70

A.1.1 HTML Beispiel . . . 70

A.1.2 CSS Beispiel . . . 72

A.1.3 SQL Beispiel . . . 73

A.2 Relevanter Code des Auditing-Systems . . . 74

A.2.1 Entität: AuditLog . . . 74

A.2.2 Serilog-Konfiguration . . . 75

A.2.3 Log-Level Enum . . . 77

A.2.4 Interceptor . . . 78

A.2.5 Registrierung: AuditingInterceptorContributor . . . 80

A.2.6 Action-Filter . . . 81

A.2.7 Event-Handler . . . 82

A.2.8 AuditingService . . . 83

A.2.9 ContextInformation . . . 88

A.2.10 AuditingHelper . . . 89

Selbstständigkeitserklärung . . . 94

Listings

2.1 Beispiel einer JavaScript-Typisierung 9

3.1 Serilog-Konfiguration innerhalb der Konfigurationsdatei „appsettings.json“ 37
3.2 Ausschnitt der AuditingService-Klasse zur manuellen Protokollierung . . . 39
3.3 Ergebnis eines manuellen Log-Eintrags im JSON-Format 40
3.4 Ergebnis eines Log-Eintrags des Interceptors im JSON-Format 41
3.5 Ergebnis eines Log-Eintrags des Event-Handlers im JSON-Format 42

4.1 Data Transfer Object für den Rückgabewert der Overview-API 46
4.2 Data Transfer Object für die Eingabeparemeter der Overview-API 46
4.3 Implementierung der Overview-API zum befüllen der Übersichtstabelle . . 47
4.4 Benutzerdefinierte Manager-Klasse mit Implementierung einer Filter-Funktion für die Übersichtstabelle 48

A.1 HTML-Code der beispielhaften Seite 70
A.2 CSS-Code der beispielhaften Webseite 72
A.3 SQL Skript zum Demonstrieren der verschiedenen SQL-Kategorien 73
A.4 Entität, welche die Tabelle in der Datenbank widerspiegelt; jede Eigenschaft inkl. Datentyp stellt eine Spalte in der Tabelle dar 74
A.5 Konfiguration des Logging-Frameworks Serilog 75
A.6 Enum zum Definieren der Wichtigkeit eines Log-Eintrags 77
A.7 Implementierung des AuditingIntercerptors 78
A.8 Implementierung des AuditingInterceptorContributor 80
A.9 Registrierung des AuditingInterceptorContributor im IoC-Container . . . 81
A.10 ActionFilter CurrentHttpContext 81
A.11 ExceptionLogger EventHandler 82
A.12 AuditingService-Klasse zum erstellen der Log-Einträge 83
A.13 ContextInformation DTO 88
A.14 Hilfsklasse AuditingHelper für übergreifende Logik 89

1 Einleitung

In datengetriebenen Webanwendungen ist das Protokollieren von Systemereignissen ein wichtiger Bestandteil. Es wird dazu verwendet, kritische Systemabläufe anhand von Fehler-, Warn- oder Informationsmeldungen zu erfassen. Dies ermöglicht den Entwicklern und administrativen Nutzern einer Plattform, den Programmverlauf detailliert zu überwachen. Besonders, wenn die Applikation einer Vielzahl von Nutzern den Zugriff auf eine Datenbank oder im Allgemeinen Datenmanipulationen erlaubt, ist eine detaillierte Protokollierung aller relevanten Aktionen des Systems essenziell.

Der Begriff „Log“ stammt aus dem Englischen und bezeichnet grundsätzlich das systematische Erfassen und Speichern von Informationen über den Programmablauf. Häufig wird das Ergebnis eines Logs, die Log-Daten, in einer Datei oder Datenbank gespeichert, sodass alle gewünschten Aktionen inklusive deren Details festgehalten werden. Läuft das System fehlerhaft oder nicht erwartungsgemäß, ist das Logging-Ergebnis ein wichtiger Anhaltspunkt für Entwickler und Administratoren, um die Ursache des Problems zu identifizieren und entsprechende Maßnahmen zu ergreifen.

Neben der Unterstützung zur Fehlersuche trägt das Logging zur allgemeinen Gesundheit des Systems und zur Transparenz von Datenänderungen bei. Durch die Implementierung eines solchen Systems hat ein Entwickler oder Administrator der Applikation im Supportfall den Zugriff auf die Aufzeichnung der Systemaktivität, was eine Unterstützung zur Lösungsfindung sein kann. Eventuelle Supportzeiten können dadurch reduziert und potenzielle Probleme frühzeitig erkannt werden. Da mit der Komplexität einer Anwendung auch die Fehleranfälligkeit steigt, nimmt der positive Einfluss eines Logging-Systems auf die Stabilität einer Anwendung entsprechend zu.

1.1 Kundenprojekt

Ein bundesweit aktiver Entsorgungsfachbetrieb mit dem Fokus auf nachhaltiges Management von Wohn- und Gewerbeimmobilien betreut aktuell rund 700.000 Haushalte. Das Unternehmen bietet verschiedene Leistungen an, wie zum Beispiel Abfallentsorgung, Bereitstellen und Zurückstellen sowie die Reinigung von Abfallbehältern.

Eine Webapplikation und eine Smartphone-App werden im Bereich der Immobilien- und Tourenverwaltung verwendet und deutschlandweit eingesetzt. Das System ist darauf ausgelegt, große Datenmengen aus Nutzerinteraktionen, Aufgabenprozessen und Datenimporten zu verarbeiten.

Das Unternehmen benötigt eine Webapplikation zur Tourenplanung und Verfolgung sowie eine Smartphone-App zur Übersicht für Liegenschaftsbetreuer. Die Webapplikation umfasst:

- Aufgabenerstellung, wie Reinigung und Behälterbereitstellung.
- Tourenplanung und Optimierung der Routen für Liegenschaftsbetreuer.
- Überwachung von Touren und Aufgabenfortschritten.

Während die Smartphone-App folgende Features unterstützt:

- Eine Tagesübersicht zur Anzeige der zu erledigenden Aufgaben und Touren.
- Eine Checkliste erledigter Aufgaben mit Dokumentation.
- Erfassen von Daten als Nachweis zur Qualitätssicherung.

1.2 Motivation und Ziele

Diese Bachelorarbeit wird in Zusammenarbeit mit der „P&M Agentur Software & Consulting GmbH“ verfasst. Im Rahmen der Arbeit wird ein Logging-System für eine bestehende Webanwendung entwickelt, das protokolliert, welche Aktionen im System vorgenommen werden. Die Webanwendung verarbeitet eine Vielzahl von Nutzeraktionen z.B. Datenänderungen, die weitere Prozesse im System beeinflussen können. Gerade in komplexen Systemen, in denen eine kleine Änderung an einem Datensatz zur Folge haben kann, dass eine geplante Tour fälschlicherweise nicht mehr stattfinden kann, gewinnt die Transparenz von Abläufen an Bedeutung. Aus diesem Grund ist es sowohl aus technischer als auch aus unternehmerischer Sicht relevant, Systemabläufe nachvollziehbar zu dokumentieren.

Aktuell fehlt der Anwendung die Protokollierung für diese Zwecke. Dadurch ist es nicht möglich, Änderungen an Daten zurückzuverfolgen. Das erschwert die Fehlersuche und die Wartung, was dazu führt, dass Probleme unaufgeklärt bleiben können und der Aufwand für deren Behebung steigt.

Ein maßgeschneidertes Logging-Modul bietet entscheidende Vorteile:

- Eine Verbesserung der Systemtransparenz durch eine geeignete Darstellung.
- Fehlerzustände können nachvollzogen werden.
- Probleme hinsichtlich der Performance können entdeckt werden.

Besonders bei einem bundesweiten Einsatz muss eine zuverlässige Plattform entwickelt werden. Daher ist gefordert, ein minimalinvasives Modul zu entwickeln, das automatisch die gewünschten Funktionen im Detail protokolliert. Gerade bei komplexen Funktionen kann es sein, dass automatisches Logging nicht genügt. Deshalb ist ebenfalls ein manueller Logging-Mechanismus zu erstellen, um einzelne Verarbeitungsschritte von Methoden dokumentieren zu können.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Technologien und theoretischen Hintergründe beschrieben, die für das Verständnis dieser Arbeit relevant sind.

2.1 Begriffsdefinition

- **Website**: Sammlung zusammenhängender Webseiten unter einer Domain.
- **Webseite**: Einzelnes Dokument einer Website.
- **Homepage**: Startseite einer Website.
- **Webbrowser**: Software (z.B. Firefox oder Chrome), die es ermöglicht, Webseiten aufzurufen und darzustellen.
- **Audit**: Systematische Prüfung von Prozessen.
- **Logging**: Aufzeichnung von Ereignissen.
- **Entität**: Eindeutig identifizierbare Einheit

2.2 Architektur von Webanwendungen

In der Softwareentwicklung wird zwischen zwei zentralen Bereichen unterschieden. Zum einen dem „Frontend“ und zum anderen dem „Backend“. Das Frontend bildet die grafische Oberfläche einer Applikation. Darunter fällt alles, was für den Nutzer sichtbar ist und womit interagiert werden kann. Das Backend hingegen definiert den Teil einer Applikation, der Funktionen und Datenoperationen erst ermöglicht. Diese Arbeit fokussiert sich auf das Front- und Backend in Bezug auf die Webentwicklung. In den folgenden

Abschnitten werden diese Ausdrücke, Frameworks sowie verwendete Programmier- und Beschreibungssprachen erklärt und zum Teil mit Anwendungsbeispielen belegt.

2.2.1 Client & Server

Die Client & Server Architektur beschreibt die Aufteilung von Webanwendungen in zwei Komponenten. Ein Teil einer Webanwendung wird auf der Seite des Nutzers ausgeführt (dem Client) und der zweite Teil wird Serverseitig ausgeführt. Ein Webserver sorgt dafür, dass die Website verfügbar ist, während der Client versucht, diese über z.B. einen Webbrowser aufzurufen. Damit ein Nutzer eine Website aufrufen kann, muss zunächst in die Adresszeile des Browsers die URL (Uniform Resource Locator) des Servers eingegeben und bestätigt werden. Im Hintergrund wird vom Webbrowser eine Anfrage über das Http-Protokoll (Abschnitt 2.4) erstellt und an den Server gesendet. Auf der Serverseite empfängt der Webserver die Anfrage und verschickt eine Antwort zurück an den Client. Der Browser empfängt diese Antwort und visualisiert die Webseite. [1]

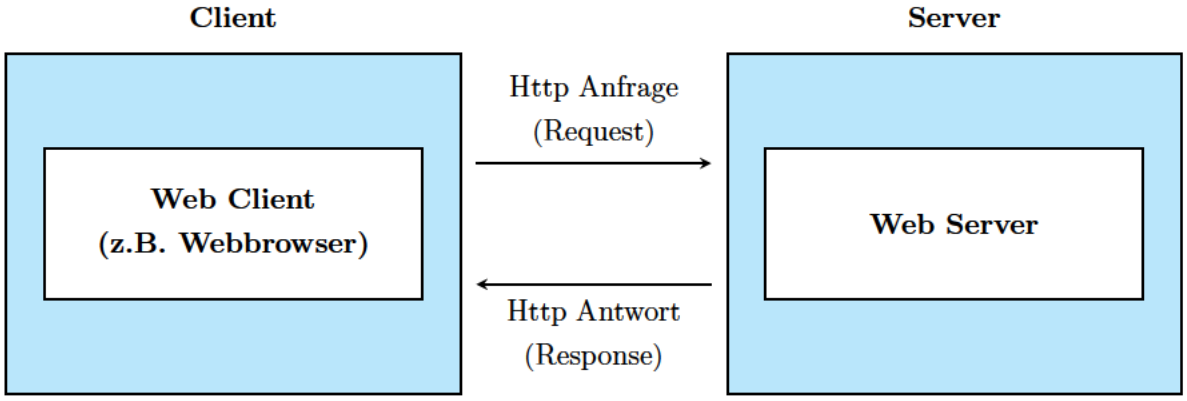

Abbildung 2.1: Kommunikation zwischen Web-Client und Web-Server in Anlehnung an [1]

Abbildung 2.1 zeigt einen Client, z.B. ein Webbrowser, der eine Http-Anfrage an den Webserver sendet. Der Server empfängt diese und liefert anschließend eine Http-Antwort zurück.

2.2.2 Frontend

Das Frontend wird als Client-Seite bezeichnet, da es im Browser des Benutzers dargestellt wird. Es beschreibt die interaktiven Komponenten einer Applikation[1]. Darunter fallen Schaltflächen, Eingabefelder oder auch die eingegebenen Daten. In der Frontend-Entwicklung ist der Hauptschwerpunkt die Benutzererfahrung, denn es werden Anwendungen erstellt, mit denen direkt interagiert wird. Dabei steht im Fokus, die Oberfläche schnell, sicher und intuitiv zu gestalten [37].

Einer der Hauptbestandteile des Frontends ist die Aufbereitung und vor allem die Visualisierung von Daten. Es geht darum, umfangreiche Datenbestände schnell und übersichtlich darzustellen. Besonders bei großen Datenmengen sind passende Darstellungsweisen, wie Tabellen oder Diagramme, zu wählen. [11]

Das Design der Komponenten einer Webseite, insbesondere die Optimierung der UI[2] (engl. User Interface) und UX[3] (engl. User Experience), sind ebenfalls ausschlaggebende Faktoren für die Benutzerzufriedenheit einer Applikation [11]. Das Design beeinflusst, wie intuitiv die Nutzung für den Anwender ist. Im Optimalfall findet sich ein Nutzer ohne weitere Informationen oder eine Einführung auf der Website zurecht und kann die gewünschten Inhalte schnell erreichen.

Hypertext Markup Language

Die Hypertext Markup Language (HTML) ist eine textbasierte Auszeichnungssprache zum Erstellen von Webseiten. Sie legt fest, welche Inhalte in welcher Reihenfolge im Browser dargestellt werden. Mit HTML ist es möglich, Texte, Bilder, Videos oder auch Verlinkungen zu anderen Webseiten einzubinden. HTML dient dabei nicht zur Gestaltung des Aussehens, sondern hauptsächlich der inhaltlichen und strukturellen Organisation durch sogenannte Tags[4].

Tags leiten HTML-Elemente ein und schließen diese auch wieder ab. Sie teilen dem Browser des Nutzers mit, welche Elemente dargestellt werden sollen und in welcher Hierarchie sich diese befinden. Tags bieten die Möglichkeit, verschiedene Inhaltsarten zu definieren,

[1] Programm, das eine bestimmte Aufgabe ausführt
[2] Benutzeroberfläche
[3] Benutzererfahrung
[4] Auszeichnung eines Elements

beispielsweise eine Überschrift (<h1> </h1> bis <h6> </h6>), Absätze ( ), Listen (<ul> , <ol>), Bilder (<img> </img>) und weiteres [1].

Die folgende Abbildung zeigt eine lokal geöffnete HTML-Seite.

Abbildung 2.2: Beispiel einer HTML-Seite

Abbildung 2.2 zeigt die vom Browser geöffnete lokale HTML-Seite „index.html“. In der Adresszeile befindet sich keine URL, sondern der absolute Pfad der Datei, da diese HTML-Seite nicht über einen Webserver bereitgestellt wird und daher nur lokal geöffnet werden kann. Da zuzüglich kein Übertragungsprotokoll angewendet wird, kann auch nicht von einer echten Webseite gesprochen werden. Die HTML-Struktur bleibt jedoch gleich.

Der Körper der Seite stellt die Überschrift, einen Absatz, eine ungeordnete Liste, das Beispielbild und die Verlinkung zur HAW-Homepage dar. Die Fußzeile wird in diesem Fall mit dem Namen des Autors befüllt. Der Code für diese Beispiel-Webseite ist als Listing

A.1 hinterlegt. Die Formatierung dieser Inhalte entsteht in Verbindung mit Cascading Style Sheets.

Cascading Style Sheets

Cascading Style Sheets (CSS) definiert das visuelle Erscheinungsbild der HTML-Elemente einer Webseite. Dazu gehören z.B. Textfarbe, Schriftgröße und Positionierung [1].

Dadurch lassen sich Formatierungen erstellen und wiederverwenden, sodass eine zentrale Anpassung für alle Elemente mit einem bestimmten Tag gilt. CSS bietet auch die Möglichkeit, individuelle Klassen zu erstellen, welche dann HTML-Elementen zugewiesen werden können. Dadurch können projektinterne Komponenten entwickelt werden, sodass ein einheitlicher Stil gilt. Es kommt des Öfteren vor, dass verschiedene Internetbrowser CSS-Code unterschiedlich interpretieren. Deshalb kann nicht uneingeschränkt davon ausgegangen werden, dass jeder Quellcode in verschiedenen Browsern identisch dargestellt wird [33].

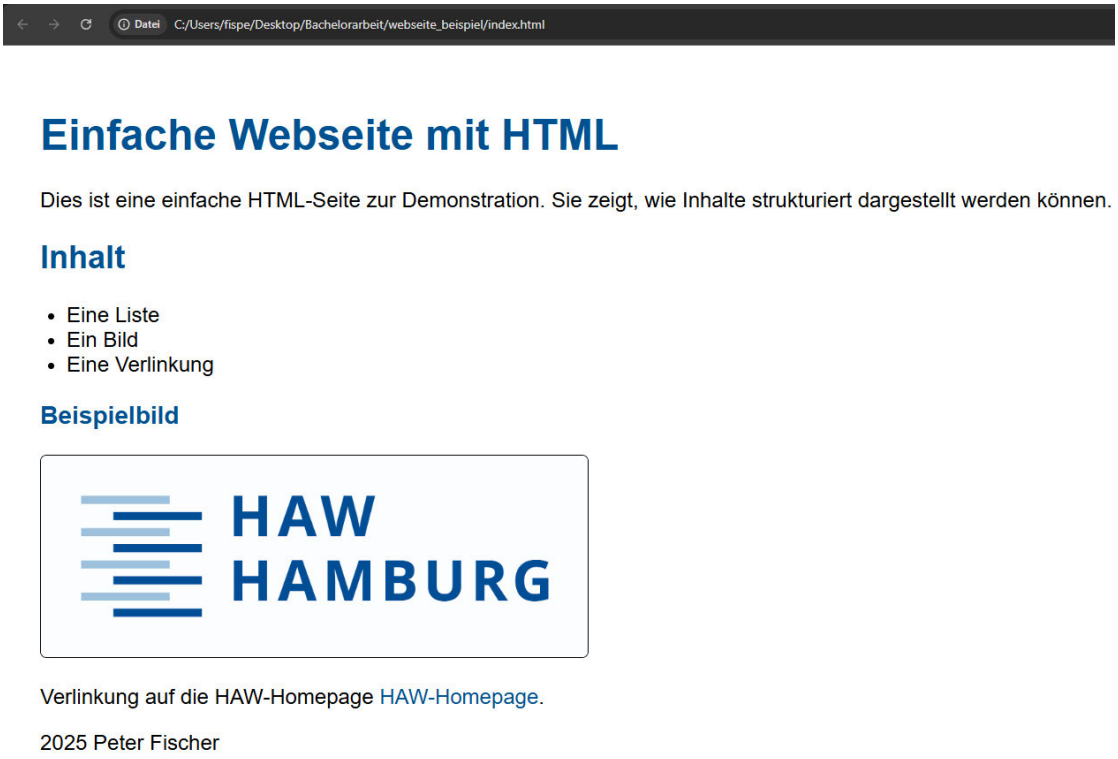

Abbildung 2.3: Beispiel einer HTML-Seite nach Ergänzung mit CSS

Nach dem Import der CSS-Datei aus A.2 wird bereits an der Schriftfarbe der Überschriften deutlich, welche Auswirkungen die Formatierungen auf die jeweiligen HTML-Elemente haben können. So wurde für das gesamte <body> HTML-Element die Schriftart auf „16px sans-serif“ gesetzt und der Abstand zu den Seitenrändern um 20 Pixel erhöht. Ebenfalls wurde ein Rahmen um das Beispielbild gezogen, um es besser hervorzuheben. Auch die Positionierung der ungeordneten Listenelemente wurde um 20 Pixel nach links verschoben.

Nachdem das Gerüst mit HTML und die Formatierung durch CSS erstellt wurden, können mithilfe von JavaScript oder TypeScript dynamische Elemente und Logik in die Webseite integriert werden.

JavaScript

Im Gegensatz zu HTML, das die Struktur einer Webseite beschreibt und CSS, welches das Design definiert, ist JavaScript eine clientseitige Programmiersprache. Sie ermöglicht es, Logik in eine Webseite zu implementieren. Mit JavaScript können interaktive Webseiten erstellt werden, die dynamisch auf Benutzeraktionen reagieren. Beispielsweise ist es möglich, Knöpfe oder sogar Spiele zu erstellen [35].

JavaScript ist eine flexible Programmiersprache, sie bietet jedoch einen großen Nachteil. Weil sie eine schwach typisierte Sprache ist, können Programmierer den Variablentyp nicht definieren. Zur Laufzeit kann eine Variable jeden beliebigen Datentyp annehmen, wodurch das Ergebnis von Operationen stark vom Datentyp der beteiligten Variablen abhängt [4].

Dadurch kann das Verhalten von Operationen zu unerwarteten Ergebnissen führen. Das zeigt folgendes Beispiel:

```
1 let a = "5";
2 let b = 5;
3                       // Ergebnis:
4 console.log(a == b); // true
5 console.log(a === b); // false
6
7 console.log("5" - 1); // 4
8 console.log("5" + 1); // "51"
```

Listing 2.1: Beispiel einer JavaScript-Typisierung

Dieses Verhalten kann unintuitiv wirken, da die Subtraktion einer Zahl mit einer Zeichenkette nicht zu einem Fehler führt, sondern JavaScript eine Typumwandlung vornimmt. Um solche Probleme abzufangen und eine strengere Typprüfung durchzuführen, wurde TypeScript entwickelt.

TypeScript

TypeScript dient als Erweiterung von JavaScript. Es ergänzt diese um Features wie die Typisierung. Sie ist eine Obermenge von JavaScript und daher abwärtskompatibel, sodass jeder gültige JavaScript-Code auch gültiger TypeScript-Code ist. Im Gegensatz zu reinem JavaScript bietet es die Möglichkeit Typen zu verwenden, wodurch Probleme bereits zur Kompilierzeit erkannt werden können. Die Typisierung ist optional, sodass flexibel zwischen typisierten und untypisierten Variablen gewählt werden kann.

TypeScript wurde eingeführt, um die Schwächen von JavaScript auszugleichen und die Stärken auszubauen. [19]

2.2.3 Backend

Die serverseitige Komponente in der Webentwicklung wird als das Backend bezeichnet. Laut [11] ist es zuständig für das Verarbeiten und Verwalten von Daten. Darunter fällt das Erstellen von Dateneinträgen, das Lesen von Informationen, das Aktualisieren von Datensätzen und deren Löschung (CRUD-Operationen). Der Begriff „CRUD“ steht für die grundlegenden Operationen mit dem Umgang von Daten: Create (Erstellen), Read (Lesen), Update (Aktualisieren) und Delete (Löschen). Das Backend stellt sicher, dass das Frontend über die notwendigen Daten verfügt, um korrekt zu funktionieren. Anders als beim Frontend, mit dem die Nutzer interagieren, arbeitet das Backend einer Webapplikation auf dem Server, in einem für den Client nicht sichtbaren Bereich.

Die Hauptaufgaben des Backends umfassen:

- **Serverseitige Logik**: Kernfunktionalität der Anwendung, Datenverarbeitung und Nutzeranfragen.
- **Performance**: Serverarchitektur, Datenbank und Backend-Logik müssen effizient arbeiten.

- **Datenbankmanagement**: Kommunikation mit den Datenbanken.
- **API-Entwicklung**: Erstellung von Schnittstellen, die den Datenaustausch zwischen Frontend und Backend ermöglichen.
- **Sicherheit**: Schutz der Anwendung durch beispielsweise Authentifizierung und Rollenverteilung.

C#

C# ist eine objektorientierte Programmiersprache mit dem Fokus auf Klassen und Objekten. „Das Konzept der objektorientierten Programmierung besteht darin, Objekte aus der realen Welt mit ihren Eigenschaften und Verhaltensweisen möglichst realitätsnah als Softwareobjekte abzubilden.“ [34, S. 45]
Das Ziel ist es, den Programmablauf so zu gestalten, wie der Prozess auch in der realen Welt ablaufen würde. Nach [24] umfasst die objektorientierte Programmierung:

- **Klassen**: Benutzerdefinierte Datentypen
- **Objekte**: Instanzen von Klassen
- **Methoden:** Funktionen innerhalb einer Klasse
- **Attribute:** Eigenschaften eines Objekts

Die Programmiersprache ist ein Teil der .NET-Plattform und dank der Integration in das Framework (Abschnitt 2.3) eignet sie sich für viele Anwendungsbereiche, sowie Desktop-Anwendnungen, Webentwicklung und Spielentwicklung [3].

2.2.4 Application Programming Interface

Eine API ist nach [13] die Art und Weise, wie verschiedene Programme oder im Allgemeinen Systeme miteinander interagieren. Sie ist eine Programmierschnittstelle und ermöglicht es, Anwendungen über eigens erstellte „Standardbefehle“ miteinander kommunizieren zu lassen. Durch diese Schnittstelle wird der Datenaustausch zwischen Systemen eindeutig definiert.

APIs sind in nahezu jedem System verbreitet und die meisten Menschen nutzen sie, ohne es zu realisieren. Jedes hochgeladene Bild oder jede Nachricht in sozialen Medien wird über eine API vom Client an den Server gesendet [36].

Sie bieten Nutzern auch die Möglichkeit, eigene Programme mit vorgefertigten Schnittstellen anderer Systeme kommunizieren zu lassen. So können beispielsweise Prozesse durch Bots automatisiert Befehle an einen Server senden.

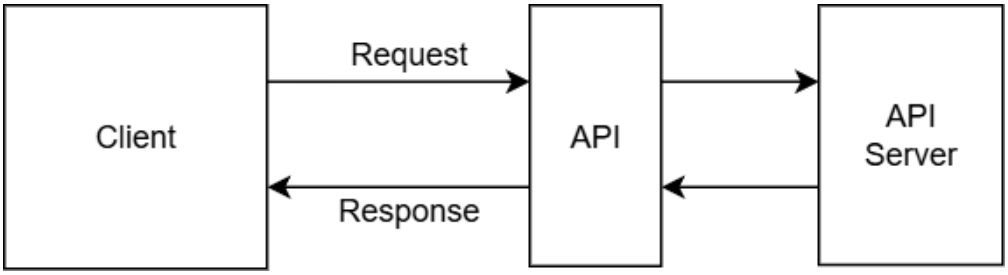

Abbildung 2.4: Austausch von Request und Response über eine API in Anlehnung an [29]

Die Abbildung 2.4 zeigt die Rolle einer API. Der Client sendet eine Anfrage an den Server, um beispielsweise einen Datensatz zu bearbeiten. Damit diese Anfrage korrekt verarbeitet werden kann, muss sie einer vordefinierten Struktur der API folgen. Diese Struktur legt fest, welche Parameter erforderlich sind (Pflichtangaben), welche optional übermittelt werden können und in welchem Format die Daten vorliegen müssen. Nur wenn die Anfrage diesen Vorgaben entspricht, kann der Server sie verarbeiten und eine entsprechende Antwort zurücksenden.

Das Arbeiten mit APIs kann jedoch eine gewisse Abhängigkeit der serverseitigen Software mit sich bringen. Änderungen an der API-Struktur können dazu führen, dass bestimmte Funktionen nicht mehr wie erwartet funktionieren oder sogar vollständig entfallen.

Data Transfer Object (DTO)

Häufig werden für den Austausch von Daten zwischen einer API und dem Server Objekte verwendet. Dadurch wird festgelegt, welche Werte und Datentypen in eine Schnittstelle hineingegeben und welche im Anschluss als Antwort zurückgesendet werden. So ist sichergestellt, dass die API nur die erforderlichen Daten akzeptiert [28].

2.3 Frameworks

Ein Framework stellt eine Art Grundgerüst für ein Softwareprojekt bereit. Es ist ein Satz aus vorgefertigten Funktionen und Bibliotheken, die die Softwareentwicklung deutlich effizienter machen. Ein großer Vorteil dessen ist, dass dadurch Programme mit weniger Fehlern innerhalb kürzerer Zeit umgesetzt werden können [14]. Durch die Verwendung von Frameworks entsteht weniger duplizierter Code, denn ein Großteil muss nicht mehr selbst implementiert werden.

2.3.1 Angular

Ein beliebtes Frontend-Framework für Webanwendungen ist Angular. Es ist ein Framework, welches für die Erstellung der Clientseite von Webapplikationen verwendet wird. Der Hauptzweck besteht darin, SPAs[5] zu entwickeln [2]. Eine Single-Page-Webanwendung wird laut [8] als eine Webapplikation definiert, welche nur ein einzelnes Webdokument lädt und dieses nur mit neuen Inhalten aktualisiert. Nach [15] ist Angular in fünf Kernkonzepte eingeteilt:

- **Komponente**: Der fundamentale Baustein einer Angular-Anwendung. Eine Komponente besteht aus einem HTML-Template, einem CSS-Style und einer TypeScript-Klasse.
- **Templates**: HTML-Dateien, die jedoch nicht nur die klassische HTML-Syntax unterstützen, sondern auch die Angular-Syntax, welche definiert, wie eine Komponente im Browser gerendert wird.
- **Direktiven**: Anweisungen innerhalb des DOM (Document Object Model), die das Verhalten oder das Aussehen von HTML-Elementen ändern können.
- **Services**: Klassen mit einem eindeutig definierten Zweck. Sie werden verwendet, um wiederholende Logik oder Daten bereitzustellen.
- **Dependency Injection**: Ein Verfahren, mit welchem eine Klasse Abhängigkeiten von anderen Klassen anfordern kann.

Die Website für das Kundenprojekt ist eine Single-Page-Application und verwendet im Frontend das Angular-Framework mit den Sprachen HTML, CSS und TypeScript.

[5] Single-Page-Applications

2.3.2 .NET (Dotnet)

.NET ist eine Softwareentwicklungsplattform, dessen Grundgedanke es ist, Anwendungen für unterschiedliche Plattformen zu entwickeln. Sie eignet sich zum Erstellen von Web-, Desktop- und Smartphone-Applikationen auf verschiedenen Betriebssystemen. Da .NET bereits seit 2002 in Entwicklung ist, sind viele Bibliotheken und Funktionen verfügbar, die es unübersichtlich machen können, die passenden Funktionen für den gewünschten Zweck zu finden. Jedoch bringt es viele erwähnenswerte Vorteile mit sich [34]:

- **Objektorientierung**: Klassen und Objekte werden grundsätzlich unterstützt.
- **Plattformunabhängigkeit**: Durch .NET können plattformunabhängige Anwendungen erstellt werden.
- **Sprachunabhängigkeit**: Es werden mehrere Programmiersprachen unterstützt, wie C# und Visual Basic.
- **Speicherverwaltung**: Durch einen Garbage-Collector werden nicht benötigte Objekte automatisch erkannt und aus dem Speicher entfernt.

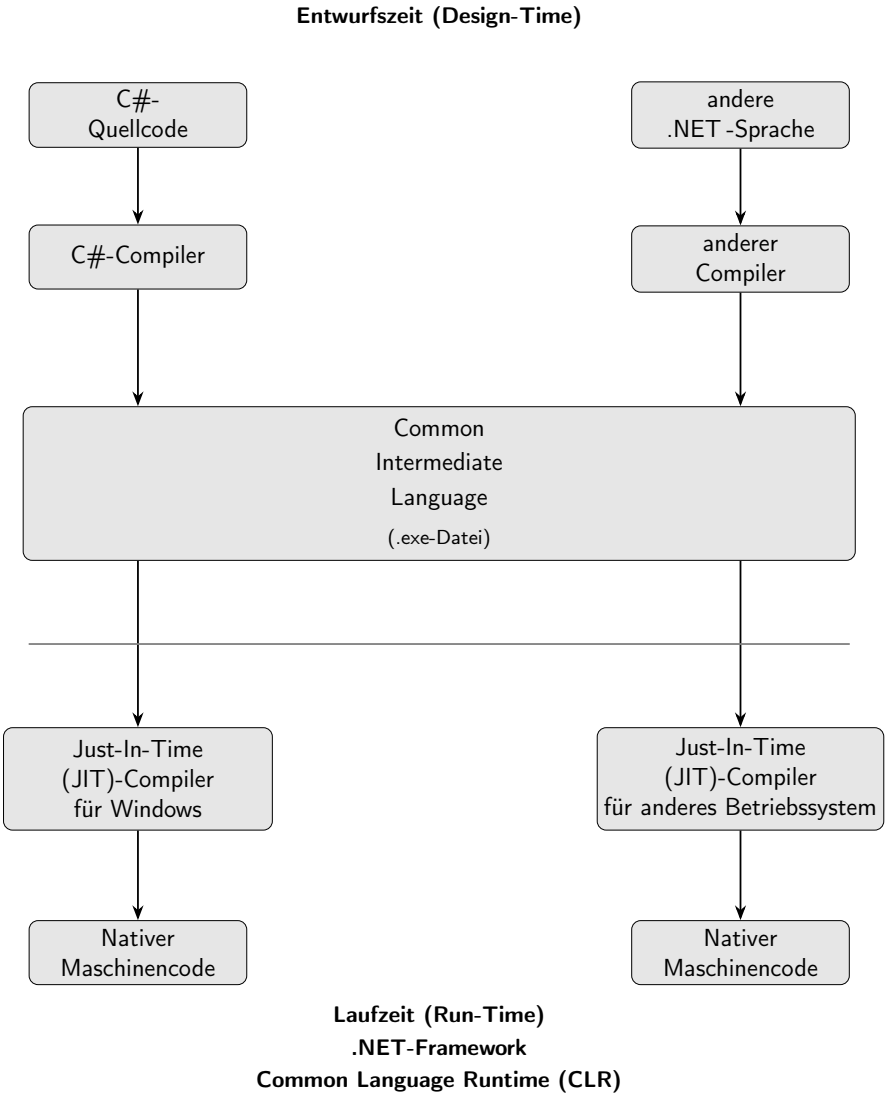

Abbildung 2.5: Kompiliervorgang der .NET-Plattform für C# und andere Sprachen in Anlehnung an [34]

Die Abbildung 2.5 zeigt den Kompiliervorgang in .NET. Zuerst wird der Quellcode einer beliebigen .NET-Sprache durch den jeweiligen Compiler in die Common Intermediate Language (CIL) übersetzt und in einer ausführbaren Datei (.exe) abgelegt. Zur Laufzeit übernimmt die Common Language Runtime (CLR) die Kompilierung und erstellt aus der CIL einen Maschinencode, der auf dem gewünschten Betriebssystem ausführbar ist.

2.3.3 Entity Framework

Das Projekt dieser Arbeit profitiert von der auf .NET basierenden Bibliothek Entity Framework, denn durch Entity Framework ist es möglich, Datenbankoperationen durch C#-Klassen anstelle von SQL-Befehlen durchzuführen. Es wird als ein Objekt-relationales Mapping (ORM)-Framework benutzt. Es bietet Entwicklern die Möglichkeit, Datenbankzugriffe über .NET-Objekte zu verwalten, anstatt SQL-Abfragen manuell zu schreiben. Die Aufgabe von Entity Framework ist es, eine einfache Verbindung zwischen dem Code und der relationalen Datenbank herzustellen [9].

2.3.4 ASP.NET Core

ASP.NET Core ist ein Framework für die Erstellung von dynamischen Webanwendungen und basiert auf der Plattform .NET. Mit diesem Framework wird das Backend der Anwendung entwickelt. Es bietet die Möglichkeit, Http-APIs zu entwickeln, die sowohl von mobilen Anwendungen als auch von Single-Page-Anwendungen genutzt werden können [25].

2.4 Hypertext Transfer Protocol

Das „Hypertext Transfer Protocol“ (HTTP) ist ein Client-Server-Webprotokoll, welches die Kommunikation zwischen Clients und Servern im Internet bestimmt. Dabei wird für jede Ressource eine separate Anfrage erstellt. Sei es beispielsweise für ein Bild oder das CSS-Style der Webseite [1].

Bei diesem Protokoll startet der Client Kommunikation mit dem Server, denn der Client stellt bei Bedarf eine Http-Request an den Server. Der Server verarbeitet diese Anfrage und sendet basierend darauf eine Antwort, die Http-Response. Dies umfasst das Lesen von Dateien, Zugriffe auf Datenbanken oder das Aufrufen von APIs. Beispielsweise werden die Argumente und die Antwort einer API über dieses Protokoll übertragen [1].

2.4.1 Status Codes

Der Statuscode einer Http-Response gibt dem Client Informationen über den Status der Anfrage. Statuscodes sind dreistellige Zahlen mit einem kurzen Beschreibungstext. Sie sind nach [7] wie folgt definiert:

- Informativ (100 - 199)
- Erfolgreich (200 - 299)
- Umleitungen (300 - 399)
- Client Fehler (400 - 499)
- Server Fehler (500 - 599)

Einige der gängigsten Codes sind in der folgenden Tabelle gelistet.

Tabelle 2.1: Übersicht gängiger Http-Statuscodes und deren Bedeutung in Anlehnung an [1]

Status Code	**Name**	**Beschreibung**
200	OK	Die Anfrage wurde erfolgreich verarbeitet
301	Moved Permanently	Die Ressource wurde permanent auf eine neue URL verschoben
400	Bad Request	Die Anfrage ist fehlerhaft oder unvollständig
401	Unauthorized	Zum Ausführen der Anfrage fehlt eine gültige Authentifizierung
403	Forbidden	Der Zugriff auf diese Ressource ist verboten
404	Not Found	Die angeforderte Ressource wurde nicht gefunden.
500	Internal Server Error	Interner Serverfehler, die Anfrage kann nicht verarbeitet werden.

Statuscodes sind besonders beim Logging hilfreich, denn die Log-Einträge können anhand dieser Codes sortiert oder gefiltert werden. Dadurch können schnell Fehlermeldungen der gleichen Art oder gravierende Fehlermeldungen wie Statuscode „500“ gefunden werden.

2.4.2 Http-Kontext

Im .NET Framework liefert der Http-Kontext alle Informationen einer Http-Anfrage und dessen Http-Antwort. Er wird initialisiert, sobald die Applikation eine Anfrage erhält [23].

Die Applikation kann auf diesen Kontext über das von ASP.NET bereitgestellte Interface `IHttpContextAccessor` zugreifen. Während dieser Kontext besteht, können daraus die übergebenen Argumente, Statuscodes, der aufgerufene Endpunkt und viele weitere Informationen extrahiert werden. Dieser Kontext bietet auch die Möglichkeit, eigene Eigenschaften zu speichern, um sie für andere Klassen zugänglich zu machen.

2.5 Relationale Datenbanken

Relationale Datenbanksysteme sind bei datengetriebenen Webanwendungen unverzichtbar. Sie speichern Daten in Form von klar strukturierten Tabellen, damit alle Einträge einer Tabelle einheitlich aufgebaut sind. Die eindeutige Identifizierung eines Datensatzes erfolgt durch einen Primärschlüssel. Mit Fremdschlüsseln können Verbindungen unter Tabellen erstellt werden, um Redundanzen[6] zu vermeiden. Durch ein Datenbank-Management-System (DBMS) können Datenbankabfragen per SQL (Abschnitt 2.5.1) oder durch eine grafische Oberfläche durchgeführt werden.

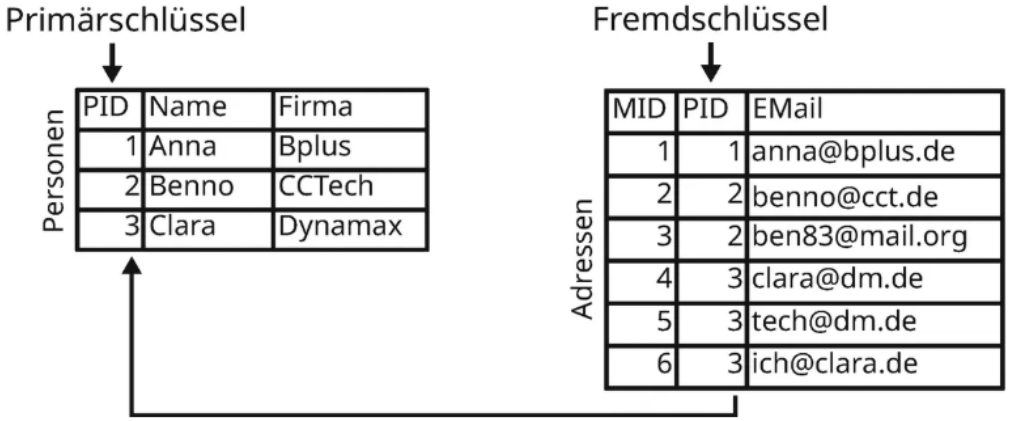

Abbildung 2.6: Verbindung von zwei Tabellen über den Primär- und Fremdschlüssel [26]

[6] Das Speichern von mehrfachen gleichen Daten

Die Abbildung 2.6 zeigt zwei relationale Datenbanktabellen (Personen und Adressen) mit deren Verknüpfung über den Primär- und Fremdschlüssel. In der linken Tabelle (Personen) wird jede Person eindeutig über den Primärschlüssel (PID) identifiziert. Neben der Personen-ID enthält die Tabelle auch die Attribute Name und Firma.

Durch den Primärschlüssel der Personentabelle und dessen Referenzierung als Fremdschlüssel in der Adresstabelle entsteht eine 1:n-Beziehung. Daher ist es möglich, eine Person mehreren E-Mail-Adressen zuzuordnen.

2.5.1 SQL-Datenbankoperationen

Die Abfragesprache SQL (Structured Query Language) zählt zu den Datenbanksprachen und dient zur strukturierten Abfrage, Erstellung oder Manipulation von Daten innerhalb relationaler Datenbanken. SQL ist nach [20] in vier Kategorien unterteilt:

- **Data Definition Language (DDL)**: Dient der Festlegung und Verwaltung der Struktur einer Datenbank, wie dem Anlegen oder Entfernen von Tabellen.
- **Data Manipulation Language (DML)**: Umfasst Befehle zur Bearbeitung der in der Datenbank gespeicherten Inhalte, wie Einfügen, Aktualisieren oder Löschen von Datensätzen.
- **Data Query Language (DQL)**: Ermöglicht die Abfrage von Informationen aus der Datenbank.
- **Data Control Language (DCL)**: Regelt Zugriffsrechte und Berechtigungen innerhalb der Datenbank.

Das im Anhang A.3 gezeigte Skript verdeutlicht, wie durch DDL eine Datenbank mit Tabelle erstellt, anhand von DML mit Daten befüllt und anschließend mittels DQL abgefragt wird. SQL bietet die Grundlage für jegliche Datenbankoperationen.

2.6 Logging

Das Logging ist der Hauptbestandteil dieser Arbeit und bezeichnet das Schreiben und Speichern von Nachrichten, die ausdrücken, was in einem Programm passiert. Es ist bedeutend für Systeme, in denen Administratoren oder Entwickler einen Einblick in die Aktionen des Programms und in die Interaktionen der Nutzer benötigen.

So ein Protokoll ist vorteilhaft für Debugging und Fehlersuche. Falls ein hartnäckiger Fehler im Programm entsteht, der nicht kurzfristig gefunden werden kann, ist es für die Entwickler essenziell herauszufinden, welcher Teil der Applikation nicht ordnungsgemäß funktioniert, um die Suche einzugrenzen. Ein Log ist daher hilfreich zum Identifizieren, durch welche Aktionen ein Fehler reproduziert werden kann oder ob es sich dabei tatsächlich um einen Fehler handelt [17].

2.6.1 Audit-Log und Debug-Log

Laut [31, S. 47] gibt es zwei Arten von Logs, die klar definierte Anwendungsbereiche haben. Zum einen Audit-Logs und zum anderen Debug-Logs. Ein Audit-Log wird benutzt, um die Aktivitäten auf der Plattform zu untersuchen. Welcher Nutzer hat welche Aktion ausgeführt oder allgemeiner: Welche Anfragen sind beim Server angekommen und sind diese erfolgreich oder fehlerhaft gewesen? In diesem Projekt wird der Audit-Log hauptsächlich von administrativen Nutzern der Applikation verwendet. Ein Debug-Log hingegen ist nur für die Entwickler einer Applikation vorgesehen. Er zeichnet auf, welche Fehler, Warnungen oder Informationsmeldungen innerhalb des Programms aufgetreten sind. Die folgende Tabelle stellt die beiden Logs gegenüber.

Tabelle 2.2: Vergleich zwischen Audit-Logs und Debug-Logs in Anlehnung an [31]

	Audit-Logs	**Debug-Logs**
Protokollierung	Immer eingeschaltet	Eingeschaltet, wenn nötig
Inhalt	Angriffe, Aktivitäten, Fehler	Fehler, Ausfälle, Störungen
Protokollierungsbereich	Im Voraus definiert	Nicht bekannt (situationsabhängig)
Zeitrahmen	Über langen Zeitraum nützlich	Für sehr kurzen Zeitraum nützlich

Die Tabelle 2.2 stellt Audit-Logs und Debug-Logs gegenüber. Audit-Logs sind hauptsächlich für Sicherheit und Nachvollziehbarkeit einer Applikation, während Debug-Logs vor allem zur Fehlerdiagnose und deren Behebung dienen. Audit-Logs sammeln dauerhaft Informationen, Debug-Logs sind zeitlich und inhaltlich begrenzt. Sie werden von Programmierern gezielt innerhalb des Codes eingesetzt, um z. B. Fehler in einem Algorithmus zu finden.

2.6.2 Speicherformate

Um Log-Einträge zu speichern, bieten sich mehrere relevante Formate an. Die am häufigsten verbreitete Art der Protokollierung ist eine textbasierte Log-Datei, da die Erstellung solcher Protokolle kostengünstig und durch die Verwendung von Frameworks vereinfacht werden kann. Textbasierte Protokolle haben auch weitere Vorteile, wie zum Beispiel, dass sie typischerweise in einem menschenlesbaren Format gespeichert sind und mit einfachem Textverarbeitungstool, wie „grep“ in Linux analysiert werden können [31, S. 74].

Für ein Projekt, in dem mehrere Benutzer gleichzeitig Log-Einträge auslösen können, trifft die Log-Datei aufgrund des gleichzeitigen Schreibzugriffs auf ihre Grenzen. In solchen Fällen ist das Datenbank-Logging geeignet, da auf eine Datenbank parallele Zugriffe möglich sind. Bei einer Vielzahl von Nutzern kann es zu großen Datenmengen kommen, sodass eine effiziente Auswertung durch SQL-Abfragen oder ein DBMS durchaus hilfreich sein kann.

Nach [31, S. 78] hat das Datenbank-Logging seine Stärken in der schnellen Durchsuchbarkeit und Filterung der Datensätze. Mithilfe von APIs können die Datensätze in einem beliebigen Format im Frontend dargestellt werden und den Einblick in Systemereignisse gewähren.

2.6.3 Logging Kriterien

Gezieltes Logging

Nach [17] sollte so viel geloggt werden wie möglich. Der tatsächliche Umfang richtet sich jedoch nach den Anforderungen der jeweiligen Anwendung. In einem Projekt, in dem ein API-Aufruf Argumente von über hunderten Objekten liefern kann, sollte nicht unbedingt jedes weitere Zwischenergebnis protokolliert werden, sondern nur das, was tatsächlichen

Mehrwert bietet. Es sollte gezielt geloggt werden, um unnötige Daten und vor allem Redundanz in einer Datenbank zu vermeiden. Was sind also Kriterien für ein gutes Logging? Laut [31, S. 47, S.332] gibt es zahlreiche Fälle, in denen Log-Einträge schlicht unvollständig oder gar nutzlos sind. Um das zu vermeiden, beschrieb er, welche Informationen für die Protokollierung sinnvoll sind, und nennt diese „the 5 W's of Logging".

- **Was ist passiert?** (Beschreibung des Ereignisses)
- **Wann ist es passiert?** (Zeitpunkt des Ereignisses)
- **Wo ist es passiert?** (Ort/Funktion des Ereignisses)
- **Wer ist beteiligt?** (Beteiligte Nutzer)
- **Woher stammt der Aufruf?** (Standort, IP-Adresse)

Genaues Logging

Logdaten werden verwendet, um Fehlerzustände zu erkennen oder um Benutzeraktionen nachzuverfolgen. Daher ist es äußerst wichtig, dass diese Daten gepflegt werden und auch genau sind. Es ist problematisch, wenn Administratoren versuchen einen Fehler zu identifizieren, der nicht reproduziert werden kann, da die Log-Einträge nicht den tatsächlichen Eingaben des Nutzers entsprechen. Kurz gesagt müssen alle Einträge frei von Fehlern oder irreführenden Informationen sein, damit das System seinen Zweck erfüllt.

2.6.4 Strukturierte Logs

Bei der strukturierten Protokollierung werden Log-Daten in einem klar definierten Format gespeichert, sodass sie leicht durchsucht, gefiltert und gegebenenfalls automatisiert weiterverarbeitet werden können. Als ein Standard für das strukturierte Logging hat sich das menschen- und maschinenlesbare Format JSON[7] durchgesetzt [32]. So können mit einem Logging-Framework, welches strukturierte Protokollierung unterstützt, gesamte Objekte als JSON serialisiert und anschließend in einer Datenbank gespeichert werden. Zu den Anwendungsfällen der Serialisierung im Bezug auf diese Arbeit gehören Übergabewerte oder Rückgabewerte von Funktionen, die strukturiert in einer Datenbank gespeichert werden können.

[7] JavaScript Object Notation

2.6.5 Unstrukturierte Logs

Im Gegensatz zum strukturierten Logging sind unstrukturierte Logs oft einfache Textdateien, die Zeichenketten und gegebenenfalls Variablen beinhalten. Sie sind in einem menschenlesbaren Format (als Text), aber für Computer schwer automatisch auszuwerten, da sie keiner definierten Struktur folgen [32].

2.6.6 Serilog

In dieser Arbeit wird das Serilog Logging-Framework für .NET-Anwendungen benutzt. Es bietet vorgefertigte Logging-Funktionalitäten, wie zum Beispiel einen Log-Kontext, über den zusätzliche Informationen an die Log-Ausgaben angehängt werden können. Dadurch können individuelle Parameter einfach hinzugefügt werden.

Sinks

Ein besonderer Vorteil von Serilog ist die Unterstützung von Sinks. Ein Sink ist dafür zuständig, Log-Daten oder Ereignisse an eine bestimmte Quelle zu senden. Dadurch können die Log-Daten in verschiedene Ziel-Systeme, wie eine Datei oder eine Datenbank, geschrieben oder sogar in Monitoring-Tools übertragen werden. Der MSSql[8]-Sink wird verwendet, um Einträge, die von Serilog mit dem Aufruf der Log-Funktion erstellt werden, in einer MSSql-Datenbank zu speichern.

[8] Microsoft SQL Server

3 Logging-System

Dieses Kapitel der Arbeit befasst sich mit der Planung und Implementierung des Logging-Systems.

3.1 Anforderungen

3.1.1 Separate Log-Datenbank

Um das Logging-System effizient zu implementieren, sodass die Performance der Webanwendung nicht wesentlich beeinträchtigt, wird eine separate Datenbank eingerichtet. Diese Datenbank beinhaltet eine Tabelle, in welcher die Log-Einträge gespeichert werden sollen. Durch diesen Ansatz wird die Hauptdatenbank der Applikation nicht zusätzlich durch das ständige Schreiben und Lesen von Log-Einträgen belastet. Somit kommen zwei voneinander getrennte Datenbanken zum Einsatz:

- Die Hauptdatenbank, in der Nutzerdaten gespeichert und alle Nutzeraktionen durchgeführt werden.
- Die Log-Datenbank, die zur Speicherung von Audit-Logs dient.

Diese Trennung stellt sicher, dass zumindest das Speichern und das Abrufen der Log-Daten keinen Einfluss auf die Performance der Hauptdatenbank haben.

3.1.2 Logging-Modul

Das Ziel dieses Moduls ist es, automatisiert ein Protokoll über die Nutzeraktionen, wie zum Beispiel das Aufrufen von datenmanipulierenden APIs, zu erstellen. Die Herausforderung dabei ist es, eine Lösung zu entwickeln, die alle Anwendungsfälle der Protokollierung abdeckt. Darunter fallen das automatische Loggen von Exceptions (Fehlern) sowie

relevanten Nutzeraktionen und die Möglichkeit für Entwickler, manuelle Log-Einträge im Code zu erstellen. Manuelle Log-Einträge sind hierbei besonders wichtig, da innerhalb von kritischen Funktionen eine Schritt-für-Schritt-Analyse der Datenänderungen erstellt wird. Dadurch kann identifiziert werden, aus welchen Gründen ein Ergebnis angezeigt wird.

An das Logging-Modul werden folgende Erwartungen gestellt:

- Automatisches protokollieren von APIs und Controllern. Neu erstellte APIs sollen ohne zusätzliche Registrierung erfasst und geloggt werden.
- Jegliche Exceptions (Fehlermeldungen), wie Authentifizierungsfehler beim fehlerhaften Login als auch benutzerdefinierte Validierungsfehler, sollen protokolliert werden.
- Manuelle Log-Einträge müssen so wenig Parameter erhalten wie nötig und so viele Informationen automatisch dem Datensatz hinzufügen wie möglich. Dadurch wird der größte Nutzen für den geringsten Aufwand sichergestellt.
- Log-Einträge sollen eindeutig identifizierbar sein.
- Nicht nur Endpunkte sollen protokolliert werden, sondern auch die von ihnen aufgerufenen Funktionen.
- Das Ergebnis (Log-Daten) soll in einem strukturierten Format dargestellt werden.
- Der bereits im Projekt global verwendete Logger (Log4Net), welcher für systemkritische Logs über das ILogger-Interface als Konsolen-Logger benutzt wird, soll unverändert bleiben. Das Logging-Modul soll unabhängig davon protokollieren.

Es ist entscheidend, dass nicht nur API-Aufrufe mit ihren Parametern protokolliert werden, sondern auch interne Funktionen mit ihren Laufzeiten, die von den APIs aufgerufen werden. Das Standard-Logging des ASP.NET Zero-Frameworks protokolliert zwar API-Aufrufe, bietet jedoch nicht die Möglichkeit, interne Funktionen ohne Codeänderungen automatisch zu protokollieren [5]. Ziel ist es, dass alle neu hinzugefügten APIs automatisch zum Logging hinzugefügt werden, ohne dass es explizit angegeben werden muss. Darüber hinaus arbeitet ein eigens entwickeltes Logging-Modul vollständig unabhängig von Frameworks und gibt den Entwicklern volle Kontrolle über Struktur, Inhalt und Eigenschaften der Log-Einträge. Aus diesen Gründen wird ein eigenes Logging-Modul konzipiert und programmiert.

Relevante Daten zur Protokollierung

Beim Protokollieren ist es essenziell, bewusst zu entscheiden, welche Informationen aufgezeichnet werden sollen und welche nicht. Speicherplatz ist eine endliche Ressource und im Falle einer Datenbank mit zusätzlichen Hosting-Kosten verbunden. Es ergibt nicht nur aus wirtschaftlicher Sicht Sinn, nur Daten mit Mehrwert zu loggen, denn das Erfassen irrelevanter Daten führt zu einem erhöhten Speicherbedarf und belastet die Performance der Datenbank. Je mehr Daten in einer Datenbank gespeichert werden, desto länger dauert die Anwendung von Such- oder Filterfunktionen.

Der Fokus liegt darauf, ausschließlich relevante Informationen zu erfassen. Relevante Informationen fallen für dieses Projekt unter die folgenden Hauptkriterien:

- **Performance-Metriken**: Startzeit, Endzeit und Ausführungsdauer
- **Parameter**: Eingabeparameter und Rückgabewerte (JSON serialisiert)
- **Kontextinformationen**: Benutzerinformationen, Sessioninformationen und Http-Request-Daten

Es kann durchaus vorkommen, dass eine API nicht mit der erwartete Geschwindigkeit ausgeführt wird. Daher müssen Performance-Metriken protokolliert werden. Strukturierte Eingabeparameter und Rückgabewerte werden protokolliert, um den Datenaustausch zwischen Client und Server oder zwischen den internen Funktionen zu beobachten. Kontextinformationen werden gespeichert, damit zurückverfolgt werden kann, welche Person für welchen Funktionsaufruf verantwortlich ist.

Die folgende Tabelle zeigt die für das Projekt relevanten Log-Eigenschaften mit einer kurzen Beschreibung.

Tabelle 3.1: Definition der Log-Eingenschaften zur Protokollierung von Ereignissen

Eigenschaft	**SQL Datentyp**	**Beschreibung**
Id	bigint	eindeutige Identifikationsnummer eines Log-Eintrags
TimeStamp	datetime2	Zeitpunkt, wann der Log-Eintrag erstellt wurde
LogLevel	int	Klassifikation für die Wichtigkeit eines Log-Eintrags
StartTime	datetime2	Startzeit der Funktion
EndTime	datetime2	Endzeit der Funktion
ElapsedMilliseconds	bigint	Ausführungsdauer der Funktion in ms
IsManual	bit	Angabe, ob manueller oder automatischer Eintrag
StatusCode	int	Statuscode einer Anfrage
Message	nvarchar	Log-Nachricht
Details	nvarchar	Details einer Exception
ExceptionType	nvarchar	Exception-Typ
Exception	nvarchar	Beschreibung einer Exception
InnerException	nvarchar	innere Exception, bei Verschachtelung
TenantId	int	Niederlassungs-ID zur Identifikation einer Niederlassung
UserId	bigint	Benutzer-ID zur Identifikation des Benutzers
UserName	nvarchar	Benutzername
ClassName	nvarchar	Klassenname, aus welcher der Log-Eintrag ausgelöst wurde
MethodName	nvarchar	Methodenname, aus welcher der Log-Eintrag ausgelöst wurde
Arguments	nvarchar	Argumente, die an eine Funktion übergeben wurden
Response	nvarchar	Rückgabewert einer Funktion
TraceId	uniqueidentifier	Trace-ID (GUID) zum Klassifizieren mehrerer Log-Einträge innerhalb eines Funktionsaufrufs

Die Tabelle 3.1 zeigt alle Log-Eigenschaften, die ein Log-Eintrag enthalten kann. Um diese Eigenschaften in einer Datenbank zu speichern, wird eine Entität erstellt (Listing A.4), die eine Tabelle der Datenbank widerspiegelt. Die Auswahl der Eigenschaften ist angelehnt an die „5W's of Logging“ nach [31]. Der letzte Punkt, **woher stammt der Aufruf** wurde jedoch nicht berücksichtigt, da dies innerhalb des Projekts nicht relevant ist. Die eindeutige Identifizierung einer Person findet durch die Tenant-ID (Niederlassungs-ID) und die User-ID (Benutzer-ID) statt, daher ist das Speichern der IP-Adresse überflüssig.

Nicht jeder Log-Eintrag muss zwingend alle diese Werte enthalten. Beispielsweise sollte es bei einem manuell erstellten Log-Eintrag nicht die Möglichkeit geben, eine Startzeit oder Endzeit zu setzen, da diese Werte keine Aussagekraft bei einem Eintrag haben, der nur zu einem bestimmten Zeitpunkt erstellt wird und keine Funktionsdauer hat. Diese leeren Felder werden mit „NULL“ befüllt. Das bedeutet, dass in einem manuellen Log-Eintrag nur vordefinierte Eigenschaften gesetzt werden dürfen.

3.2 Konzept

Das Grundkonzept besteht aus einem automatischem Logging für APIs und registrierte Funktionen und Exceptions sowie zusätzlich ein manuelles Logging bei Bedarf. Um das automatische Logging umzusetzen stehen die folgenden Ansätze zur Auswahl:

- **Source Code Generation**
- **Interceptor**
- **ActionFilter**
- **EventHandler**

Im Folgenden werden diese Begriffe erklärt und mit ihren Vor- und Nachteilen gegenübergestellt. Das Ziel ist es durch die Auswahl der genannten Möglichkeiten und einer eventuellen Kombination dieser Ansätze eine vollständige Protokollierung gemäß Tabelle 3.1 zu erreichen. Dabei soll der vorhandene Quellcode nicht modifiziert oder beeinträchtigt werden.

3.2.1 Source Code Generation

Source Code Generation bedeutet, dass zur Kompilierzeit des Programms zusätzlicher Code generiert wird [6]. Dieser wird zum bestehenden Code hinzugefügt und kann eine Logging-Funktionalität unterstützen. Der Vorteil dabei liegt darin, dass kein zusätzlicher Code in vorhandene Dateien geschrieben, sondern „nur“ der Generator konfiguriert werden muss. Allerdings ist ein Nachteil und der Grund, warum sich gegen dieses System entschieden wurde, dass der generierte Code schlecht wartbar ist. Er wird erst zur Kompilierzeit generiert und ist daher für den Entwickler in der Datei nicht sichtbar.

3.2.2 Interceptors

Interceptors dagegen arbeiten zur Laufzeit des Programms. Ein Interceptor kann durch die Implementierung des `IInterceptor`-Interfaces des .NET-Frameworks erstellt werden. Es bietet die Funktion `public void Intercept(IInvocation invocation)`, mit der Möglichkeit, Aufrufe abzufangen, bevor oder nachdem diese ausgeführt werden [10]. Ein korrekt konfigurierter Interceptor kann also beim Aufruf einer registrierten Funktion die `Intercept`-Methode ausführen, welche nach Belieben programmiert werden kann. Ein entscheidender Vorteil dieser Lösung ist die geringe Invasivität. Bestehender Code muss nicht verändert werden, denn der Interceptor läuft parallel dazu. Alle relevanten Daten zum Funktionsaufruf werden als `IInvocation`-Objekt übergeben.

3.2.3 Action-Filter

Action-Filter lassen sich durch das `IActionFilter`-Interface des ASP.NET-Frameworks implementieren. Ein Action-Filter erlaubt es, verschiedene Aktionen (z.B. Http-Aufrufe), die von der Anwendung verarbeitet werden, zu überprüfen und zu bearbeiten. Action-Filter bieten Methoden, die vor oder nach dem Aufruf einer Aktion ausgeführt werden, um individuelles Verhalten hinzuzufügen. Dabei können Informationen wie Http-Methode, Statuscode, Argumente und Rückgabewerte gelesen oder neue Attribute zum Http-Kontext hinzugefügt werden. Alle Klassen, die Zugriff auf diesen Kontext haben, können auf die hinterlegten Eigenschaften des Action-Filters zugreifen [39].

3.2.4 Event-Handler

Event-Handler in .NET erlauben eine Registrierung von Subscribern (Abonnenten) bei einem Provider (Anbieter), um eine Benachrichtigung zu erhalten, sobald ein bestimmtes Event eingetreten ist. Tritt ein Ereignis ein, sendet der Provider eine Benachrichtigung an die Empfänger, welche dann eine entsprechende Reaktion definieren. [21]

3.3 Implementierung

Der folgende Abschnitt beschreibt die Umsetzung des Logging-Moduls in Bezug auf die genannten Mechanismen. Es wird beschrieben, wie sie in die Applikation implementiert werden, welche Aufgabenbereiche sie übernehmen und wie durch eine Kombination einiger Ansätze eine flexible Protokollierung entsteht.

3.3.1 Architektur

Das Logging-Modul wurde durch eine Kombination aus **Interceptor, Action-Filter und Event-Handler** umgesetzt. Diese Ansätze ergänzen sich, da sie unterschiedliche Bereiche der Applikation abdecken und gemeinsam eine vollständige Protokollierung ermöglichen. Im Folgenden werden die Einsatzgebiete der verschiedenen Technologien beschrieben.

Interceptor

Der Interceptor wird im Projekt dazu verwendet, um automatisch alle Funktionen der registrierten Klassen zu unterbrechen und die Intercept-Funktion auszuführen. Innerhalb dieser Funktion wird das übergebene Objekt `IInvocation invocation` gelesen. Dieser Parameter enthält aufrufbezogene Daten, wie Funktionsname, Klassenname oder Argumente der API-Funktion [10]. Die relevanten Daten werden anschließend verarbeitet und als ein neues Objekt an die Logging-Funktion übergeben, die basierend auf den vorhandenen Daten einen Log-Eintrag erstellt.

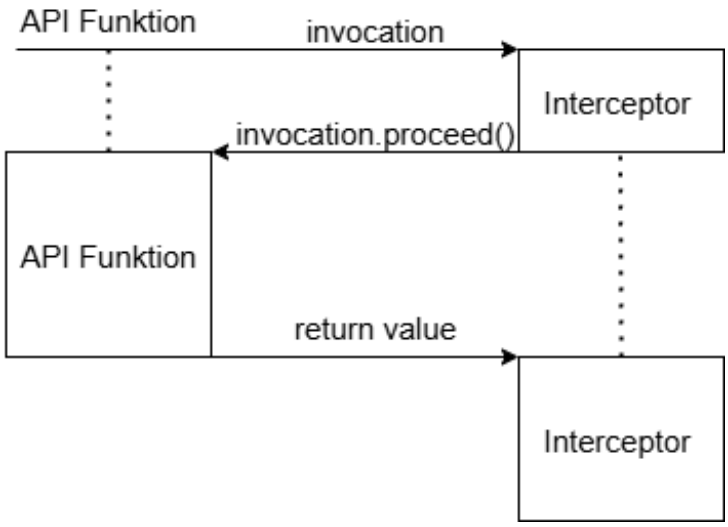

Abbildung 3.1: Ablauf zwischen einer abgefangenen Funktion und dem Interceptor

Abbildung 3.1 veranschaulicht den schematischen Ablauf des Interceptors. Zu Beginn wird eine Funktion aufgerufen, die abgefangen werden soll. Beispielsweise eine API-Funktion. Diese Funktion wird unterbrochen und das `IInvocation`-Objekt an den Interceptor überreicht. Dieser führt seine Funktion aus, jedoch ist zu beachten, dass die API-Funktion gestoppt bleibt, bis in der Interceptor-Klasse `invocation.proceed()` aufgerufen wird. Nach Abschluss der API wird der Rückgabewert erfasst und als Eigenschaft des Objekts `IInvocation invocation` gespeichert. Auf diese Weise können Eigenschaften, wie Argumente, Rückgabewerte, Klassenname, Methodenname und viele weitere, an eine Logging-Funktion übergeben werden.

Der zugehörige kommentierte Quellcode des Interceptors ist in Listing A.7 dargestellt.

Action-Filter

In der Applikation wird ein ASP.NET Core Action-Filter verwendet, um zusätzliche Logik vor der Ausführung einer Http-Aktion zu implementieren. Das `IActionFilter`-Interface stellt die folgenden Methoden:

- **OnActionExecuting(ActionExecutingContext context)**: Diese Funktion wird vor Aktion ausgeführt. Hier können eingehende Parameter gelesen oder modifiziert werden.

- **OnActionExecuted(ActionExecutingContext context)**: Ist eine Methode, die nach Ausführung der Aktion aufgerufen wird. Hier ist es möglich, z. B. Rückgabewerte zu modifizieren [22], jedoch wird das in dieser Arbeit nicht benötigt.

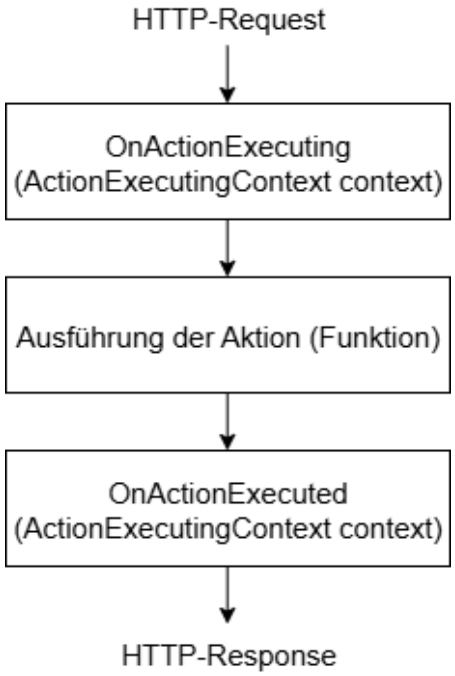

Abbildung 3.2: Ablaufdiagramm des Action-Filters während einer Http-Anfrage

Die Abbildung 3.2 zeigt: Sobald eine Methode aufgerufen wird, die eine Http-Anfrage erzeugt, wird vor der Ausführung zuerst der Action-Filter aktiv. Dort können beispielsweise alle Eingabeparameter gelesen oder individuelle Informationen innerhalb dieses Http-Aufrufs gespeichert werden. Erst danach erfolgt der eigentliche Methodenaufruf [39].

In dieser Arbeit wird ein Action-Filter verwendet, um eine Trace-ID und die Argumente zu speichern. Die Argumente sind wichtig für die Implementierung des Event-Handlers. Die Trace-ID wird verwendet, um einen Eintrag oder eine Gruppe zusammenhängender Einträge eindeutig über denselben Kontext identifizieren zu können.

Der Unterschied zum Interceptor ist, dass dieser auf beliebige Methodenaufrufe angewendet werden kann und die Ausführung der Funktionen unterbricht. Dahingegen ist

ein Action-Filter speziell nur für die Http-Kommunikation implementiert. Der kommentierte Code der Klasse `CurrentHttpContext`, welche das `IActionFilter`-Interface implementiert, befindet sich in Listing A.10.

Event-Handler

Angenommen, der Benutzer möchte in der Webapplikation eine neue Person anlegen, gibt aber eine ungültige E-Mail-Adresse an. Im Normalfall sollte das nicht vorkommen, da das Frontend die Eingabe validieren muss, um eine Anfrage an das Backend zu schicken. Kommt dies dennoch vor, weil im Frontend nicht korrekt validiert wird, sollte das Logging-System diesen Fehler erkennen. Das ASP.NET Framework bietet eine Klasse `ValidationAttribute`, womit Validierungen für Eingabeparameter erstellt werden können. Schlägt nun solch eine Validierung in einem API-Aufruf fehl, wird die Aktion von ASP.NET abgebrochen, bevor der Interceptor greifen kann. Das führt dazu, dass in diesem Fall kein Log-Eintrag erstellt werden kann. Um eine vollständige Protokollierung zu ermöglichen, muss ein Event-Handler für den Typ <ExceptionData> eingeführt werden.

Das Ereignismodell in .NET basiert auf einem Delegaten, der ein Ereignis mit seinem Handler verbindet. Ein Delegat ist ein Datentyp, der auf eine Methode verweist, die die Parameterliste und den Rückgabewert für ein Ereignis bereitstellt. Der Event-Handler-Delegate ist eine Methode, die beim Eintreten des Ereignisses aufgerufen wird und die Daten des zugewiesenen Datentyps entgegennimmt [21]. `ExceptionData` ist eine Klasse, welche von `EventData` erbt und daher dem Event-Handler zugewiesen werden kann. Sobald der Event-Handler ein Objekt von Typ `ExceptionData` erstellt, wird die Funktion `HandleEvent(ExceptionData eventData)` ausgeführt.

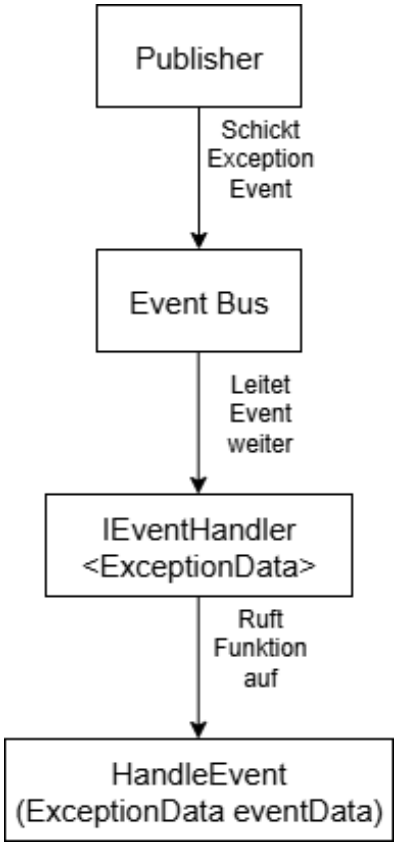

Abbildung 3.3: Ablaufdiagramm des Event-Handlers am Beispiel von `<ExceptionData>`

Die Abbildung 3.3 zeigt die Aufrufpipeline des Event-Handlers. Sobald ein Event eintritt, sendet der Publisher die Event-Daten an den ABP (ASP.NET Boilerplate) Event-Bus. Dieser leitet die Daten an alle registrierten `IEventHandler<T>`-Implementierungen weiter. Sobald ein Handler Daten erhält, wird die `HandleEvent()`-Funktion aufgerufen. In diesem Fall werden in der `HandleEvent`-Funktion die Exception-Daten extrahiert, in einem DTO gespeichert und dann als ein Objekt an die Logging-Funktion übergeben. Die Implementierung des Event-Handlers befindet sich in Listing A.11.

3.3.2 Hilfsklassen

Im Folgenden werden die eingesetzten Hilfsklassen beschrieben. Sie implementieren die eigentliche Logging-Funktionalität und befüllen die Kontextinformationen wie Benutzer- und Mandanten-IDs, Statuscodes sowie Methoden- und Klassennamen. Darüber hinaus übernehmen sie die Serialisierung von Argumenten und Rückgabewerten, damit manuelle als auch automatische Log-Einträge einheitlich gespeichert werden können.

AuditingService

Dies ist das Kernstück der Protokollierung. In dieser Klasse werden die Methoden zum Erzeugen von Log-Einträgen implementiert. Dabei wird für jeden Mechanismus (**Interceptor, EventHandler** und **manuelle Einträge**) eine eigene Log-Funktion erstellt. Das ist nötig, da der Event-Handler eine andere Teilmenge der benötigten Parameter übergeben bekommt, als der Interceptor und manuelle Log-Einträge ebenfalls nur einen Bruchteil aller vorhandenen Eigenschaften enthalten dürfen. In diesen Funktionen werden die Eigenschaften individuell befüllt, sodass eine einheitliche Protokollierung gegeben ist.

Beispielsweise bekommt der Interceptor (AuditingInterceptor-Klasse) den für das Logging relevanten Klassen- und Methodennamen im `IInvocation`-Objekt zur Verfügung gestellt, der Event-Handler (ExceptionLogger-Klasse) aber nicht. Aus diesem Grund müssen die Parameter für das Protokollieren von Exceptions aus dem Http-Kontext entnommen werden, welcher zuvor vom Action-Filter (CurrentHttpContext-Klasse) befüllt wurde. Eine weitere Besonderheit ist, dass die `<ExceptionData>` des Event-Handlers auch keine Eingabeparameter enthält. Das bedeutet, sobald eine Exception in einer API auftritt, können die Argumente nicht eingesehen werden. Das ist jedoch bei einem Validierungsfehler notwendig, um herauszufinden, welche Eingabe den Fehler verursacht.

Durch des zuvor beschriebenen Action-Filters werden die Eingabeparameter einer Http-Anfrage in dessen Kontext gespeichert und sind damit innerhalb der Event-Handlers verfügbar. So entsteht durch die Kombination zweier Mechaniken ein vollständiger Log-Eintrag zu einer Exception, auch wenn diese nicht vom Interceptor abgefangen werden konnte.

Durch Dependency-Injection kann der `AuditingService` in beliebige Klassen injiziert und seine Funktionen verwendet werden. Listing A.12 stellt die kommentierte `AuditingService`-Klasse dar.

AuditingHelper

Die `AuditingHelper`-Klasse übernimmt die Aufbereitung der Log-Einträge mit den notwendigen Eigenschaften. Sie stellt Hilfsmethoden bereit, um Informationen aus dem aktuellen Http-Kontext oder der aktiven Sitzung aus einem `IAbpSession`-Objekt zu extrahieren. Auf diese Weise kann für die Mechanismen **Interceptor** und **Event-Handler** sichergestellt werden, dass Benutzername, Mandanten-ID, Trace-ID, Statuscode sowie Klassen- und Methodennamen, sofern vorhanden, gesetzt sind.

Zusätzlich implementiert der `AuditingHelper` die Logik zur Serialisierung von Argumenten und Rückgabewerten, sodass diese in strukturierter Form gespeichert werden können. Listing A.14 zeigt die kommentierte `AuditingHelper`-Klasse.

Registrierung des Interceptors

Die Registrierung des Interceptors erfolgt in zwei Schritten. Zuerst wird der Interceptor den betroffenen Klassen zugewiesen, sodass er bei deren Aufrufen berücksichtigt wird. Dazu wird die Klasse `AuditingInterceptorContributor` erstellt, die diese Zuordnung übernimmt und den Interceptor einbindet. Zum anderen ist es erforderlich, die Klasse `AuditingInterceptorContributor` im Inversion-of-Control-Container (IoC-Container) zu registrieren, damit die Registrierung berücksichtigt wird.

Inversion of Control (IoC) ist ein technisches Prinzip, bei dem die Verantwortung für das Erstellen von Objekten nicht mehr bei der Klasse liegt, sondern an einen externen IoC-Container abgegeben wird. Die Klassen bekommen ihre Abhängigkeiten vom Container bereitgestellt [30].

Der Code aus Listing A.8 gewährt einen Einblick, wie die Registrierung im Programm stattfindet.

Serilog Konfiguration

Um Serilog gemäß den Anforderungen nutzen zu können, muss das Logging-Framework über die .NET-Konfigurationsdatei „appsettings.json“ eingerichtet werden. In diesem Projekt schreibt Serilog mithilfe des `MSSqlServer`-Sinks direkt in die separate Logging-Datenbank.

Der folgende Ausschnitt der Serilog-Konfiguration zeigt, wie die Verbindung zwischen dem Logging-Modul und der MSSQL-Datenbank konfiguriert wird.

```
"Serilog": {
  "Using": [ "Serilog.Sinks.MSSqlServer" ],
  "MinimumLevel": {
    "Default": "Information",
    "Override": {
      "Microsoft": "Fatal",
      "System": "Fatal",
      "Quartz": "Fatal"
    }
  },
  "WriteTo": [
    {
      "Name": "MSSqlServer",
      "Args": {
        "connectionString": "PmAuditLogDb",
        "tableName": "AuditLogs",
        "autoCreateSqlTable": false,
        "columnOptionsSection": {
          ...
        },
        "columnOptions": {
          ...
        }
      }
    }
  ],
}
```

Listing 3.1: Serilog-Konfiguration innerhalb der Konfigurationsdatei „appsettings.json“

Listing A.5 zeigt die Konfigurationsdatei für das Serilog-Logging-Framework. Zuerst wird MSSqlServer als Sink registriert und das minimale Log-Level auf Informations-Logs gesetzt. Weil nur fatale Fehler von Microsoft, des Systems und von Quartz geloggt werden sollen, werden die minimalen Log-Level auf „Fatal“ gesetzt. Die Verbindung zur Datenbank erfolgt über einen Connection-String, eine Zeichenkette, welche die notwendigen Parameter zur Verbindung enthält. Darunter die Adresse des Datenbankservers, den Namen der Datenbank, den Benutzernamen und das Passwort.

Da die Datenbank und die Tabelle unabhängig von Serilog erstellt werden, wird der Parameter `autoCreateSqlTable` deaktiviert. `columnOptionsSection` gibt an, welche Standardspalten entfernt und welche Spalten hinzugefügt werden. Selbst wenn Serilog die Datenbank nicht selbst erstellt, benötigt es die Informationen, welche Spalten zur Verfügung stehen, um die Einträge korrekt in der Tabelle zu speichern. Die vollständige Konfiguration ist in Listing A.5 dargestellt.

3.4 Validierung

Zur Validierung des Logging-Moduls wurde überprüft, inwiefern die in den vorherigen Abschnitten definierten Anforderungen erfüllt wurden. Hierbei steht die Frage im Vordergrund, ob die erarbeiteten Methoden für **Interceptor, Event-Handler, Action-Filter** und das Erstellen **manueller Log-Einträge** in der Praxis zuverlässig sind.

3.4.1 Manuelle Log-Einträge

Zuerst wird geprüft, ob manuelle Einträge mit minimalem Aufwand erstellt werden können und dennoch vollständig gespeichert werden. Die Entwickler sollen so wenige Angaben machen müssen wie möglich, während das Logging-Modul das Speichern von Kontextinformationen automatisiert. Diese Funktionalität wird durch die Hilfsklasse `AuditingHelper` unterstützt und die Klasse `AuditingService` bietet die Funktionen `CreateInfoLog`, `CreateDebugLog`, `CreateWarnLog` und `CreateErrorLog`. Diese können aufgerufen werden, um einen manuellen Log-Eintrag zu erstellen. Der einzige Parameter, der dafür benötigt wird, ist die Log-Nachricht. Das Protokollieren von Argumenten ist nicht immer notwendig, daher sind diese nur optional. Die Quelle des Aufrufs (Dateipfad/Dateiname und Funktionsname) wird durch spezielle Attribute `[CallerMemberName]` und `[CallerFilePath]` automatisch gesetzt (Listing A.12).

Die genannten Funktionen des Services setzen zum Schluss das Log-Level und leiten die Information an die generische Funktion `CreateLog` weiter, welche den Log-Kontext befüllt und den Eintrag erstellt.

```
public class AuditingService : ISingletonDependency
{
   // Manuellen Info-Log schreiben
   public void CreateInfoLog(
      string message,
      object arguments = null,
      [CallerMemberName] string callerMethod = "",
      [CallerFilePath] string callerPath = "")
   {
      CreateLog(LogLevel.Information, message, arguments,
         callerMethod, callerPath);
   }

   // Gemeinsame Log-Methode fuer alle manuellen Logs
   private void CreateLog(
      LogLevel level,
      string message,
      object arguments = null,
      string callerMethod = "",
      string callerPath = "")
   {
      Action<Exception, string> logAction = AuditingHelper
         .GetLogAction(level);
      LogContextInformation li = new LogContextInformation()
      {
         IsManual = true,
         LogLevel = level,
         ContextInformation = _auditingHelper
            .GetContextInformation(callerMethod, callerPath),
         Arguments = AuditingHelper.SerializeOrNull(arguments),
         Message = message
      };
      CreateLogContext(li, logAction);
   }
   ...
}
```

Listing 3.2: Ausschnitt der AuditingService-Klasse zur manuellen Protokollierung

Der obige Codeausschnitt zeigt, wie das DTO eines manuellen Eintrags durch die Funktion `GetContextInformation(callerMethod, callerPath)` mit Kontextinformationen befüllt und an die Funktion `CreateLogContext(li, logAction)` zum Protokollieren übergeben wird. Das folgende Listing zeigt den Datensatz eines manuellen Log-Eintrags im JSON-Format.


```
{
      "Id" : 246,
      "TimeStamp" : "2025-09-03T17:36:51.677Z",
      "LogLevel" : 1,
      "StartTime" : null,
      "EndTime" : null,
      "ElapsedMilliseconds" : null,
      "IsManual" : 1,
      "StatusCode" : 200,
      "Message" : "Manual log test",
      "Details" : null,
      "ExceptionType" : null,
      "Exception" : null,
      "InnerException" : null,
      "TenantId" : 2,
      "UserId" : 3,
      "UserName" : "admin",
      "ClassName" : "CareTakerAppService",
      "MethodName" : "Create",
      "Arguments" : "{\"ZipCode\":null,\"City\":null,\"FirstName\":\"
         Peter\",\"LastName\":\"Fischer\",\"Street\":null,\"
         HouseNumber\":null,\"PhoneNumber\":\"123456\",\"Email\":\"
         peter.fischer@fischer.de\",\"CompanyId\":235,\"Remarks\":
         null}",
      "Response" : null,
      "TraceId" : "A1D9E7BE-42DC-4F04-8C8E-DC5C705103B4"
      }
```


Listing 3.3: Ergebnis eines manuellen Log-Eintrags im JSON-Format

Der gesamte Log-Eintrag aus Listing 3.3 wurde durch die folgende Zeile erstellt, wobei `filterInput` der Eingabeparameter einer Overview-Methode ist:
`_auditingService.CreateInfoLog("Manual log Test", filterInput);`

3.4.2 Interceptor Log-Einträge

Der Interceptor fängt automatisch alle Funktionen der registrierten Klassen ab und führt die `Intercept(IInvocation invocation)`-Funktion aus. In dieser Funktion wird das `createInput`-DTO befüllt, welches dann die Funktion zum Erstellen des Log-Eintrags `CreateInterceptorLog(createInput)` der `AuditingService`-Klasse übergeben wird. Durch das Implementieren einer Blacklist für Klassen und Methoden können diese gezielt vom Logging ausgeschlossen werden.


```
{
      "Id" : 247,
      "TimeStamp" : "2025-09-03T17:36:53.016Z",
      "LogLevel" : 1,
      "StartTime" : "2025-09-03T17:36:51.558Z",
      "EndTime" : "2025-09-03T17:36:52.917Z",
      "ElapsedMilliseconds" : 1359,
      "IsManual" : 0,
      "StatusCode" : 200,
      "Message" : "Completed",
      "Details" : null,
      "ExceptionType" : null,
      "Exception" : null,
      "InnerException" : null,
      "TenantId" : 2,
      "UserId" : 3,
      "UserName" : "admin",
      "ClassName" : "CareTakerAppService",
      "MethodName" : "Create",
      "Arguments" : "{\"input\":{\"ZipCode\":null,\"City\":null,\"
         FirstName\":\"Peter\",\"LastName\":\"Fischer\",\"Street\":
         null,\"HouseNumber\":null,\"PhoneNumber\":\"123456\",\"Email
         \":\"peter.fischer@fischer.de\",\"CompanyId\":235,\"Remarks
         \":null}}",
      "Response" : "{\"CreatorUserName\":\"admin admin\",\"
         LastModifierUserName\":null,\"CreationTime\":\"2025-09-03T17
         :36:51.7246523Z\",\"LastModificationTime\":null,(...)}",
      "TraceId" : "A1D9E7BE-42DC-4F04-8C8E-DC5C705103B4"
      }
```


Listing 3.4: Ergebnis eines Log-Eintrags des Interceptors im JSON-Format

Listing 3.4 zeigt einen automatisch erstellten Eintrag des Interceptors im JSON-Format. Da eine abgefangene Funktion eine Laufzeit hat, sind hier im Gegensatz zum manuellen Log die Start- und Endzeit sowie die Verarbeitungsdauer gesetzt. Diese API liefert einen Rückgabewert zurück an den Client, welcher als `Response` gespeichert ist. Zusätzlich wurde erfolgreich validiert, dass die Trace-ID des Interceptor-Aufrufs mit der des manuellen Eintrags übereinstimmt. Dadurch kann der manuelle Log-Eintrag eindeutig einem API-Aufruf zugeordnet werden, da pro API-Aufruf nur ein automatischer Eintrag ($IsManual = 0$) erzeugt wird.

3.4.3 Exception Log-Einträge

Exceptions werden durch den ABP-Event-Handler abgefangen. Das `createInput`-DTO wird durch das `ExceptionData`-Objekt aus der `Exception Logger`-Klasse mit Exception-Daten und Kontextinformationen befüllt. Die Eingabeparameter der fehlgeschlagenen Funktionen werden von diesem Objekt nicht bereitgestellt. Für diesen Fall wurde der Action-Filter entworfen, der die Argumente eines Http-Aufrufs im Http-Kontext speichert und somit für andere Klassen zugänglich macht. Zuletzt wird das DTO der Logging-Funktion übergeben und ein Eintrag erstellt.


```
{
   "Id" : 248,
   "TimeStamp" : "2025-09-03T17:40:04.725Z",
   "LogLevel" : 8,
   "StartTime" : null,
   "EndTime" : null,
   "ElapsedMilliseconds" : null,
   "IsManual" : 0,
   "StatusCode" : 400,
   "Message" : "Method arguments are not valid! See ValidationErrors
      for details.",
   "Details" : "The Email format is incorrect.",
   "ExceptionType" : "AbpValidationException",
   "Exception" : "Abp.Runtime.Validation.AbpValidationException:
      Method arguments are not valid! See ValidationErrors for
      details.(...)",
   "InnerException" : null,
   "TenantId" : 2,
   "UserId" : 3,
```

```
   "UserName" : "admin",
   "ClassName" : "CareTakerAppService",
   "MethodName" : "Create",
   "Arguments" : "{\"input\":{\"ZipCode\":null,\"City\":null,\"
      FirstName\":\"Peter\",\"LastName\":\"Fischer\",\"Street\":null
      ,\"HouseNumber\":null,\"PhoneNumber\":\"12345\",\"Email\":\"
      peter.fischer@fischer.-de\",\"CompanyId\":235,\"Remarks\":null
      }}",
   "Response" : null,
   "TraceId" : "ABEEE461-AA73-4A84-A598-9748763A2B19"
}
```


Listing 3.5: Ergebnis eines Log-Eintrags des Event-Handlers im JSON-Format

Der Datensatz aus Listing 3.5 zeigt einen Log-Eintrag, der durch das Auslösen einer Exception erstellt wurde. Im Gegensatz zu den vorherigen zwei Beispielen werden hier zusätzlich Fehlerinformationen `Exception`, `ExceptionType` und gegebenenfalls `InnerException` protokolliert. Laut Fehlermeldung entspricht die E-Mail-Eingabe nicht dem erwarteten Format. Verglichen mit dem Datensatz aus Listing 3.4, in welchem die gleiche API erfolgreich war, ist erkennbar, dass die E-Mail-Eingabe sich leicht unterscheidet. Ein „-“ in der Top-Level-Domain (.de) erzeugt einen Fehler bei der Validierung der Eingabe.

4 API-Entwurf

Dieses Kapitel handelt vom Entwurf der Schnittstellen für das Auditing-System. Das Ziel ist es, APIs zu erstellen, die die Daten für das Frontend des Audit-Logs und zukünftige Statistik-Daten bereitstellen. Bei den API-Schnittstellen des Auditing-Systems steht im Vordergrund, dass diese leicht erweiterbar und effizient sind. Alle Schnittstellen dieses Kapitels sind ausschließlich zum Lesen von Daten, denn das Schreiben der Log-Einträge übernimmt das Serilog-Framework (innerhalb der `AuditingService`-Klasse). Alle Eingaben und Rückgaben erfolgen als ein Data Transfer Object, damit das Frontend Informationen darüber bekommt, welche Daten gesetzt werden können und wie die Rückgabe strukturiert ist.

4.1 Welche Schnittstellen sind notwendig?

Welche APIs benötigt werden und wie sie aufgebaut sind, ist abhängig davon, welche Daten im Frontend angezeigt werden sollen. Zuerst wird die Hauptschnittstelle entworfen, die als `Overview` bezeichnet wird. Diese bietet eine Übersicht von allen Log-Einträgen in der Datenbank und wird dazu verwendet, die Übersichtstabelle im Frontend zu befüllen. Ein Nutzer übergibt dafür Filterdaten (z.B. Zeitraum, Statuscode, Benutzername etc.) in die `Overview`-Funktion und erhält als Resultat eine gefilterte und sortierte Liste aller Einträge. Das Frontend ist anschließend dafür verantwortlich, die erhaltenen Daten in eine Tabelle zu übertragen oder anderen Komponenten zuzuordnen.

Ergänzend zur `Overview`-API, welche für eine schnelle Übersicht der Datensätze konzipiert ist und daher nur die wichtigsten Informationen (z.B. Log-Level, Statuscode, Niederlassungs-ID, Benutzer-ID etc.) zurückgibt, braucht es noch einen weiteren Endpunkt. Einen, der detaillierte Informationen zu einem Eintrag liefert.

Des Weiteren werden zusätzliche APIs benötigt, die eine zukünftige spezifische Auswertung ermöglichen. So ergeben sich die folgenden Schnittstellen für das Auditing-System:

- **Overview**: Liefert eine Übersicht aller Log-Einträge anhand von Filterparametern
- **GetDetails**: Bietet eine detaillierte Ansicht zu einem Log-Eintrag, einschließlich Argumenten, Rückgabewerten und weiteren relevanten Parametern.
- **GetMostUsedEndpoints** liefert eine Übersicht über die am häufigsten aufgerufenen Endpunkte.
- **GetCommonExceptions** stellt alle Fehlermeldungen mit Fehlerquelle sowie Häufigkeit dar und ermöglicht so die Identifikation wiederkehrender Probleme.
- **GetEndpointInfo**: gibt Kennzahlen zu einem Endpunkt zurück, darunter die Anzahl der erfolgreichen und fehlgeschlagenen Aufrufe sowie minimale, maximale und durchschnittliche Antwortzeiten

4.2 Implementierung einer Overview-API

In diesem Projekt werden APIs über die spezielle Klasse `ApplicationService` des ASP.NET Boilerplate Frameworks bereitgestellt. Sie werden benutzt, um Backend-Funktionen im Frontend zur Verfügung zu stellen. Zuerst wird ein DTO erstellt, welches das Ergebnis der API widerspiegelt. Um eine neue Schnittstelle für das Auditing-System zu entwickeln, wird eine separate Klasse erstellt, die die API-Funktionen implementiert. Im folgenden Beispiel wird die Klasse `AuditingAppService` benannt. Sie erbt sowohl von dem Interface `IAuditingAppService`, das die zu implementierenden Funktionen als eine Art Vertrag vorgibt, als auch von der Klasse `ApplicationService`, durch die die Funktionen dem Frontend erst bereitgestellt werden können.

Nach [18] werden die `ApplicationService`-Methoden aus dem Frontend aufgerufen. Dabei werden als Ein- und Ausgabewerte typischerweise DTOs verwendet, um Datenstrukturen eindeutig zu definieren. Der große Vorteil dabei ist, dass Frontend-Entwickler nur die Daten bekommen, die sie benötigen. Die folgende Klasse `AuditingOverviewDto` repräsentiert das DTO, welches an das Frontend gesendet wird.

```
public class AuditingOverviewDto : EntityDto<long>
{
    public LogLevel LogLevel { get; set; }
    public DateTime TimeStamp { get; set; }
    public int? TenantId { get; set; }
    public long? UserId { get; set; }
    public string UserName { get; set; }
    public string ClassName { get; set; }
    public string MethodName { get; set; }
    public int StatusCode { get; set; }
}
```

Listing 4.1: Data Transfer Object für den Rückgabewert der Overview-API

Es ist zu beachten, dass keine unnötigen Informationen versendet werden, sondern nur die Informationen, welche vom Frontend für eine Übersicht der Log-Einträge benötigt werden. Als Nächstes wird das DTO für die Eingabeparameter definiert. Es beinhaltet alle filterbaren Eigenschaften, nach denen sich die Übersicht richtet.

```
public class AuditingOverviewInputDto : IFilterInput
{
    public LogLevel LogLevel { get; set; }
    public DateTime? StartTime { get; set; }
    public DateTime? EndTime { get; set; }
    public ICollection<int> TenantIds { get; set; }
    public long? UserId { get; set; }
    public string UserName { get; set; }
    public string ClassName { get; set; }
    public string MethodName { get; set; }
    public ICollection<int> StatusCodes { get; set; }
    public ICollection<int> Ids { get; set; }
    public int MaxResultCount { get; set; }
    public int SkipCount { get; set; }
    public string Sorting { get; set; }
}
```

Listing 4.2: Data Transfer Object für die Eingabeparemeter der Overview-API

Listing 4.2 definiert die filterbaren Eigenschaften. Nachdem die Objekte zur Ein- und Ausgabe definiert wurden, folgt die Implementierung der Schnittstelle `Overview(AuditingOverviewInputDto input)`, die den Datenbankzugriff über diese Objekte ermöglicht. Für eine Übersicht müssen alle Datensätze geladen und danach

die Filter angewendet werden. Der folgende Codeausschnitt zeigt die Implementierung dieser Funktion.

```
1  // Zugriff auf diese API nur fuer Nutzer mit bestimmter Berechtigung
2  [AbpAuthorize(AppPermissionsGenerated.Pages_Administration_Auditing)]
3  public class AuditingAppService : ApplicationService, IAuditingAppService
4  {
5     // Asynchrone API zur Verarbeitung und Rueckgabe der gefilterten Logs
6     public Task<PagedResultDto<AuditingOverviewDto>> Overview(
          AuditingOverviewInputDto input)
7     {
8        // Setzt die TenantId als Filter, sofern vorhanden
9        if (AbpSession.TenantId.HasValue)
10       {
11          input.TenantIds = new List<int> { AbpSession.TenantId.Value };
12       }
13
14       // Mappt die Eingaben auf die Klasse AuditingFilterInput und speichert
             das Filtererebnis als AuditLog in query
15       IQueryable<AuditLog> query = _manager.Filter(ObjectMapper.Map<
             AuditingFilterInput>(input));
16
17       // Standard Sortierung, falls keine gesetzt ist
18       if (input.Sorting.IsNullOrEmpty())
19       {
20          input.Sorting = "TimeStamp desc";
21       }
22
23       int totalCount = query.Count();
24
25       // Sortieren der Abfrage (query)
26       IQueryable<AuditLog> orderedAndPagedQuery = query
27          .OrderBy(input.Sorting)
28          .PageBy(input);
29
30       // Speichern der Abfrage als Liste
31       List<AuditLog> pagedList = orderedAndPagedQuery.ToList();
32
33       // Mappen der Datenbank-Entitaeten in die DTO Struktur fuer die Ausgabe
34       List<AuditingOverviewDto> mappedDtos = ObjectMapper.Map<List<
             AuditingOverviewDto>>(pagedList);
35
36
37       return Task.FromResult(new PagedResultDto<AuditingOverviewDto>(
```

```
38          totalCount,
39          mappedDtos
40      ));
41   }
42 }
```

Listing 4.3: Implementierung der Overview-API zum befüllen der Übersichtstabelle

In Listing 4.3 werden die Daten innerhalb der API verarbeitet und die Datenbankabfrage als ein IQueryable[1]-Objekt gespeichert. Über das Attribut `[AbpAuthorize]` wird der Zugriff nur für berechtigte Benutzer (Host-Admin und Niederlassungs-Admin) ermöglicht. Zuerst wird der Input (`AuditingOverviewInputDto`) auf den benutzerdefinierten Datentypen (`AuditingFilterInput`) abgebildet. Das ist erforderlich, da die Filter-Funktion einen anderen Datentyp erwartet, als in die Overview-Funktion gegeben wird, da der Filter generisch implementiert werden soll. Wenn die Eigenschaften beider DTOs gleich benannt sind, ist nach einer Registrierung in einem „Custom-Mapper“ ein automatisches DTO-Mapping möglich. Nach dem Aufrufen der Filterfunktion der benutzerdefinierten Klasse `AuditingManager` wird die Abfrage gespeichert und im Anschluss nach dem Erstellungszeitpunkt sortiert und paginiert. Zum Schluss wird die Abfrage als eine Liste gespeichert und auf das Ausgabe-DTO gemappt. Die folgende Manager-Klasse beinhaltet die Filterfunktion.

```
1  // Manager-Klasse mit Singleton-Abhaengigkeit
2  public class AuditingManager: ITransientDependency
3  {
4     // Funktion zum Filtern von Audit-Logs nach den Kriterien im
          AuditingFilterInput
5     public IQueryable<AuditLog>Filter(AuditingFilterInput input)
6     {
7        // Benutzerdefinierte Funktionen zum setzen der Start- und Endzeit, um
             ohne Zeitangabe im Filter nur Logs der letzten 24 Stunden
             anzuzeigen
8        input.StartTime = AdjustStartTime(input.StartTime);
9        input.EndTime = AdjustEndTime(input.StartTime.Value, input.EndTime);
10
11       // Alle Log-Eintraege holen
12       IQueryable<AuditLog> query = GetAll();
13
14       // Erstellung der Abfrage mit Anwendung der Filter
15       query = query
```

[1] Schnittstelle im .NET-Framework, die eine Datenbankabfrage darstellt

```
16          // Pflichtfilter
17          .Where(e =>
18             e.IsManual == input.IsManual)
19          .Where(e =>
20             e.TimeStamp >= input.StartTime && e.TimeStamp <= input.EndTime)
21          // Optionale Filter
22          .WhereIf(input.LogLevel != LogLevel.All,
23             e => e.LogLevel == input.LogLevel)
24          .WhereIf(!input.TenantIds.IsNullOrEmpty(),
25             e => e.TenantId.HasValue && input.TenantIds.Contains(e.TenantId.
                   Value))
26          .WhereIf(!input.ClassName.IsNullOrEmpty(),
27             e => e.ClassName.ToLower().Contains(input.ClassName.ToLower()))
28          .WhereIf(!input.MethodName.IsNullOrEmpty(),
29             e => e.MethodName.ToLower().Contains(input.MethodName.ToLower()))
30          .WhereIf(!input.UserName.IsNullOrEmpty(),
31             e => e.UserName.ToLower().Contains(input.UserName.ToLower()))
32          .WhereIf(input.UserId.HasValue,
33             e => e.UserId.HasValue && e.UserId.Value == input.UserId.Value)
34          .WhereIf(!input.StatusCodes.IsNullOrEmpty(),
35             e => e.StatusCode.HasValue && input.StatusCodes.Contains(e.
                   StatusCode.Value))
36          .WhereIf(!input.Message.IsNullOrEmpty(),
37             e => e.Message.ToLower().Contains(input.Message.ToLower()))
38          .WhereIf(input.MinDurationMilliseconds.HasValue,
39             e => e.ElapsedMilliseconds.Value >= input.MinDurationMilliseconds
                   .Value);
40       // Rueckgabe der Abfrage nach Anwendung der Filter
41       return query;
42    }
43 }
```

Listing 4.4: Benutzerdefinierte Manager-Klasse mit Implementierung einer Filter-Funktion für die Übersichtstabelle

Listing 4.4 zeigt die Manager-Klasse, welche durch die Funktion `Filter(AuditingFilterInput input)`, die filternde Logik für die Datenbankabfrage bereitstellt. Ihr wird die Abhängigkeit `ITransientDependency` versehen. Im ABP-Framework gibt es die Auswahl zwischen **Scoped-**, **Transient-** und **Singleton-**Dependencies [16], welche dem IoC-Container definieren, wie die Klassen instanziiert werden sollen. Nach [27] sind die Abhängigkeiten wie folgt beschrieben:

Eine **Singleton**-Abhängigkeit bedeutet, dass nur eine Instanz dieser Klasse in der gesamten Applikation erstellt wird. Sie eignet sich für zustandslose, globale Services.

Durch **Transient** wird für jeden Aufruf eines Services eine neue Instanz dieser Klasse erstellt. Es ist die sauberste Implementierung, da keine Klasse wiederverwertet oder zwischengespeichert wird. Sie eignen sich gut für „leichte“ und zustandslose Services.

Die **Scoped**-Abhängigkeit sorgt dafür, dass für jeden neuen Http-Aufruf eine neue Instanz erstellt und innerhalb dessen wiederverwendet wird. Der Vorteil ist es, dass ein Zustand innerhalb eines Aufrufs erhalten bleibt und danach verworfen wird. Dadurch kann kein Datenleck zwischen APIs entstehen.

Für die Klasse `AuditingManager` wurde die `ITransientDependency`-Abhängigkeit gewählt, da dies eine leichtgewichtige, zustandslose Klasse ist, die dynamisch gebaut und verworfen werden kann.

Am Beispiel der Filter-Funktion wird deutlich, wie Datenbankabfragen mithilfe von Entity Framework über C#-Objekte anstelle eines SQL-Skripts erstellt werden können. Zu Beginn werden die Start- und Endzeit gesetzt, sofern diese nicht im `input` definiert sind. Ohne dieses zwingende Setzen würden alle Log-Einträge im Backend abgerufen werden, wenn nicht explizit nach einem Zeitraum gefilter wird. Da die Log-Einträge über einen längeren Zeitraum gespeichert werden sollen, wäre es dadurch theoretisch möglich, eine unbegrenzte Anzahl an Log-Einträgen abzurufen. Bei großen Datenmengen kann sich dadurch die Antwort einer API verzögern und das wirkt sich beim initialen Laden negativ auf die Benutzererfahrung aus. Im Anschluss werden alle Log-Einträge als eine Abfragequelle gespeichert und die Filter angewendet. Zuerst die verpflichtenden Filter, sowie `IsManual`, welches standardmäßig auf `false` gesetzt ist, da die Übersicht nur automatisch erstellte Log-Einträge darstellen soll, sowie der Filter nach `TimeStamp`, der den Zeitraum angibt. Zum Schluss werden die optionalen Filter gesetzt, sofern vorhanden. Zurückgegeben wird die vom `input` abhängige Abfragequelle `query`.

5 Visualisierung

Dieser Teil der Arbeit handelt von der Darstellung der Log-Daten im Frontend der Applikation. Das Ziel ist es, den Nutzern eine übersichtliche und intuitive Plattform zur Verfügung zu stellen, um effizient und gezielt nach Log-Einträgen suchen und filtern zu können. Das Frontend für das Logging-System ist in zwei Abschnitte gegliedert. Zum einen die Log-Übersicht und zum anderen die Detailansicht.

Da diese Applikation von mehreren Niederlassungen genutzt wird, welche keine Einsicht in die Log-Daten anderer Niederlassungen bekommen dürfen, muss eine Lösung implementiert werden, welches die Auditing-Oberfläche abhängig des angemeldeten Benutzers anzeigt. Ein Host-Admin[1] muss die Möglichkeit haben, die Log-Einträge aller Niederlassungen zu sehen und nach ihnen zu filtern. Ein Tenant-Admin[2] darf nur die Logs seiner eigenen Niederlassung sehen. Es ist zu beachten, dass die gesamte Auditing-Seite nur für administrative Nutzer sichtbar ist.

Da im Frontend auf viele verschiedene Bereiche geachtet werden muss, sowie die Erstellung der Tabellenstruktur und Filter, Routing, API-Aufrufe, Datenverwaltung und vieles Weitere, wird sich in diesem Kapitel nicht auf den Quellcode, sondern lediglich auf die Darstellung und Benutzererfahrung fokussiert. Da dies ein bereits fortgeschrittenes Projekt ist, sind einige Funktionalitäten, wie das Routing, bereits gegeben, welche dementsprechend nur um die neue Auditing-Seite ergänzt werden müssen.

[1] Höchste administrative Rolle allgemein und besitzt keine Niederlassung
[2] Höchste administrative Rolle innerhalb einer Niederlassung

5.1 Log-Übersicht

Die Übersicht soll dem Nutzer einen schnellen Überblick über die Log-Daten liefern. Hier werden alle verfügbaren Logs in tabellarischer Form mit einer Eingabemöglichkeit für Filter angezeigt. Da in dieser Ansicht nur begrenzt Platz zur Verfügung steht, muss im Vorfeld entschieden werden, welche Informationen relevant sind, um einen gesuchten Log-Eintrag erfolgreich zu identifizieren. Zur Auswahl stehen alle Eigenschaften aus der Tabelle 3.1. Aus diesen Eigenschaften wurden die folgenden ausgewählt, die auch als Filtereingaben fungieren:

- **Log-Level**: zum Einordnen der Wichtigkeit des Eintrags.
- **Zeitpunkt**: Zeitstempel des Ereignisses, um Logs zeitlich anzuordnen.
- **Klassenname**: die Klasse im Code, aus der das Ereignis stammt, um die Herkunft des Eintrags zu bestimmen.
- **Methodenname**: die Methode im Code, um die Herkunft zu bestimmen.
- **Statuscode**: um Details zur Abfrage zur Verfügung zu stellen.
- **Niederlassungs-ID (Tenant-ID)**: zum Einordnen, welche Niederlassung diesen Eintrag verursacht hat.
- **Benutzer-ID** : zum Einordnen, welcher Benutzer diesen Eintrag verursacht hat.
- **Benutzername**: zur Einfachheit für Administratoren, die sofort den Namen sehen, ohne in der Datenbank nach der ID zu suchen.

Die folgenden Abbildungen zeigen die Auditing-Übersicht. Zuerst die Variationen der Filter für einen Host-Admin und einen Tenant-Admin, sowie im Anschluss die Tabelle mit den Log-Einträgen.

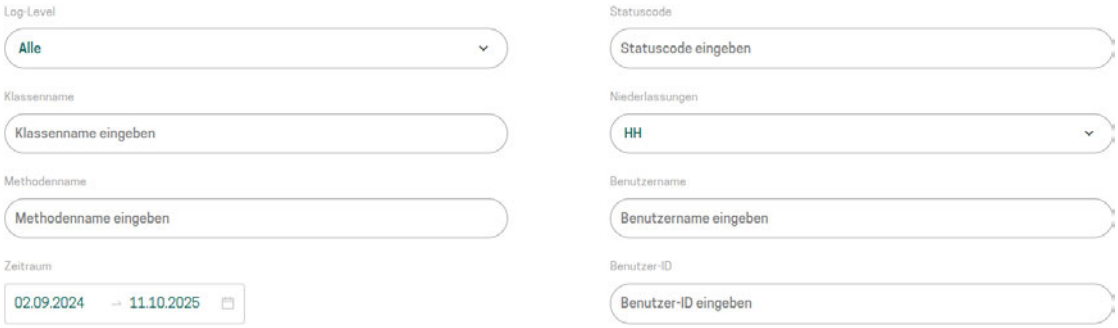

Abbildung 5.1: Darstellung der Auditing-Filter für die Rolle des Host-Admins

Abbildung 5.1 zeigt die Filtermöglichkeiten eines Host-Admins. Dieser kann nach Niederlassungen filtern, ein Tenant-Admin jedoch nicht. Wann dieses Dropdown angezeigt werden soll, ist abhängig von der Rolle des angemeldeten Benutzers. Um die bedingte Darstellung zu ermöglichen, bieten sich zwei essenzielle Parameter an, die `Tenant-ID` und die `Position-ID`. Die `Position-ID` gibt Auskunft über die Rolle einer Person. Ist dieser Wert „0“, so ist diese Person ein administrativer Nutzer. Wenn zusätzlich die `Tenant-ID` gesetzt ist, ist dieser Nutzer ein Tenant-Admin. Ist diese aber leer, dann ist der Nutzer ein Host-Admin. Sobald der Nutzer eine Tenant-ID besitzt, wird diese automatisch in die `Overview`-API als Filtereingabe übergeben und der Niederlassungs-Filter ausgeblendet. Die folgende Abbildung zeigt die Übersichtstabelle, in die das Ergebnis der `Overview`-API gemappt wird.

Zeitpunkt	Klassenname	Methodenname	Statuscode	TenantId	UserId	Benutzername
2025-08-27T12:33:35.2...	LocationAppService	ValidLocationAddresses	200	4	3	admin
2025-08-27T12:33:34.8...	CoworkerAppService	Overview	200	4	3	admin
2025-08-27T12:33:34.1...	CareTakerAppService	Overview	200	4	3	admin
2025-08-27T12:33:33.5...	CoworkerAppService	Overview	200	2	3	admin
2025-08-27T12:33:33.3...	TaskAppService	Overview	200	2	3	admin
2025-08-27T12:33:32.9...	LocationAppService	DataCollectionOverview	200	2	3	admin
2025-08-27T12:33:31.1...	DashboardAppService	GetAdditionalTasksOver...	200	2	3	admin

< 1 ... 47 48 49 50 51 > 10 / Seite

Abbildung 5.2: Darstellung der Übersichtstabelle für Log-Einträge

Abbildung 5.1 zeigt die zuvor bestimmten Eigenschaften als Spalten der Tabelle. Dadurch können die Benutzer dieses Logging-Systems schnell nach dem Status des Logs

(Information, Debug, Warnung, Fehler), Quelle oder Benutzerinformationen filtern. Das „Drei-Punkte-Menü“ auf der rechten Seite öffnet die Detailansicht zu einem Eintrag.

5.2 Log-Detailansicht

Die Detail-Seite dient dazu, jede Eigenschaft aus 3.1 darzustellen. Sie besteht aus zwei Bereichen. Zum einen der Tabelle über die verwandten Log-Einträge, welche dem Benutzer die manuellen Logs darstellt, die innerhalb des Aufrufs eines automatischen Logs erstellt wurden. Zum anderen der Detailansicht, welche alle Eigenschaften mit ihren Werten darstellt

Die Detailansicht ist das zentrale Element der Log-Darstellung. Ohne sie wäre jegliche vorherige Arbeit nahezu sinnfrei, da die gewonnene Information nicht dargestellt werden könnte. Sie trägt damit wesentlich zur Transparenz der Applikation bei. Die folgende Abbildung zeigt die Tabelle, die alle verwandten Log-Einträge beinhaltet. Einträge sind verwandt, wenn sie die gleiche Trace-ID teilen.

	Automatisch	Zeitpunkt	Klassenname	Methodenname	Statuscode	TenantId	UserId	Benutzername
ⓘ	Ja	2025-09-03T21:02:16....	CareTakerAppSer...	Overview	200	2	3	admin
ⓘ	Nein	2025-09-03T21:02:16....	CareTakerAppSer...	Overview	200	2	3	admin

Abbildung 5.3: Darstellung für verwandte Log-Einträge

Abbildung 5.3 zeigt die Tabelle, welche neben dem API-Aufruf die manuellen Einträge innerhalb desselben Http-Aufrufs anzeigt. Der Interceptor ruft die `CreateInterceptorLog`-Funktion erst nach dem vollständigen Durchlauf der abgefangenen Funktion auf. Da manuelle Logs innerhalb der abzufangenden Funktion erstellt werden, wird der Log des automatischen „Haupt-Aufrufs“ immer zuletzt geschrieben. Sortiert man die Einträge nach der Zeit absteigend, ist der „Haupt-Aufruf“ immer der erste Eintrag der Tabelle. Durch das Klicken des „Drei-Punkte-Menüs“ eines Eintrags, werden dessen Details in die Informationsansicht gemappt. Die nächste Abbildung stellt die Informationsansicht der Eigenschaften dar.

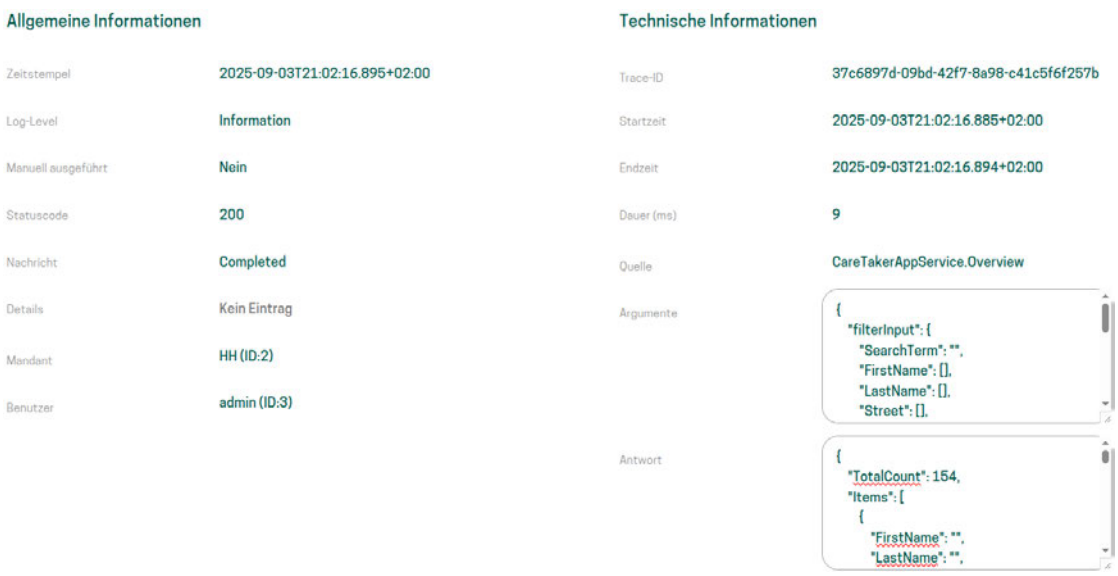

Abbildung 5.4: Darstellung der Informationsansicht eines Log-Eintrags; Exception und Exceptiontyp sind nicht aufgrund von besserer Lesbarkeit nicht abgebildet

Die Informationsansicht aus Abbildung 5.4 stellt sämtliche verfügbaren Informationen zu einem Log-Eintrag bereit. Diese ist in zwei Spalten gegliedert: „Allgemeine Informationen“ und „Technische Informationen“. Die allgemeinen Informationen sind für Administratoren ausgelegt, welche nicht zwingend einen Einblick in die technischen Details sowie Exceptions benötigen. Die technischen Informationen, wie zum Beispiel Argumente oder Funktionsdauer, helfen Entwicklern, Bugs oder allgemein Fehler zu erkennen und nachzuverfolgen.

6 Log-Analyse

Log-Daten lassen sich auf verschiedene Arten analysieren. Zum einen können die Daten manuell und zum anderen auch automatisiert durch spezielle oder selbst erstellte Analysetools durchsucht werden. Eine manuelle Durchsuchung eignet sich, wenn der Benutzer eine konkrete Hypothese hat oder nach bereits bekannten Fehlern sucht. Das ist besonders im Kontext dieses Projekts wichtig, weil Bugs in Funktionen, welche Business-Logik beinhalten, vom Logging-System nicht als Fehler wahrgenommen werden können. Daher müssen Logs von komplizierten und verschachtelten Funktionen im Bereich der Tourenplanung und Verwaltung manuell durchsucht werden, sobald den Entwicklern ein Bug auffällt. Dadurch, dass die Log-Daten in einer Datenbank gespeichert werden und dementsprechend viele Datensätze durchsucht werden müssen, stößt die manuelle Analyse für das gesamte System schnell an ihre Grenzen. Nach [31, S. 134] sind einige dieser Grenzen:

- Manuelle Analyse wird mit steigender Anzahl an Einträgen ineffizienter.
- Manuelle Analyse ist nur bei selbsterklärenden Logs hilfreich. Da das Auditing-Modul alle wichtigen Kontextinformationen und Parameter aufzeigt, trifft dieser Nachteil in dieser Arbeit nicht zu.
- Manuelle Analysen werden vermutlich niemals einen Überblick über das gesamte System verschaffen können.

In solchen Fällen kann eine automatische Untersuchung maßgeblich zur Gesundheit einer Applikation beitragen. In dieser Arbeit wird ein einfacher „Outlier-Detection“ (Ausreißererkennung) Algorithmus für eine Laufzeitanalyse eingeführt, um einen Grundstein für künftige automatische Analysemöglichkeiten zu schaffen.

6.1 Outlier

Nach [12] wird ein „Outlier“ als eine Anomalie, also ein Datenpunkt, beschrieben, der sich von anderen Beobachtungen innerhalb eines Datensatzes unterscheidet. Sie können durch Messfehler, seltene Ereignisse oder durch fehlerhaften Code entstehen und dadurch einen großen Einfluss auf statistische Analysen haben.

6.2 Automatische Analysemöglichkeiten

Es gibt zahlreiche Algorithmen, die eine Analyse von Log-Daten ermöglichen. Da innerhalb der Log-Eigenschaften nur die Funktionsdauer `ElapsedMilliseconds` ein metrischer Wert ist, bietet sich eine Laufzeitanalyse an. Zwei einfache Analysemöglichkeiten sind die Methode zur Standardabweichung und der Interquartilsabstand (IQR). Zur Darstellung dient ein Datensatz von 50 Einträgen einer `Overview`-API. Die folgende Abbildung zeigt die Werteverteilung des Datensatzes.

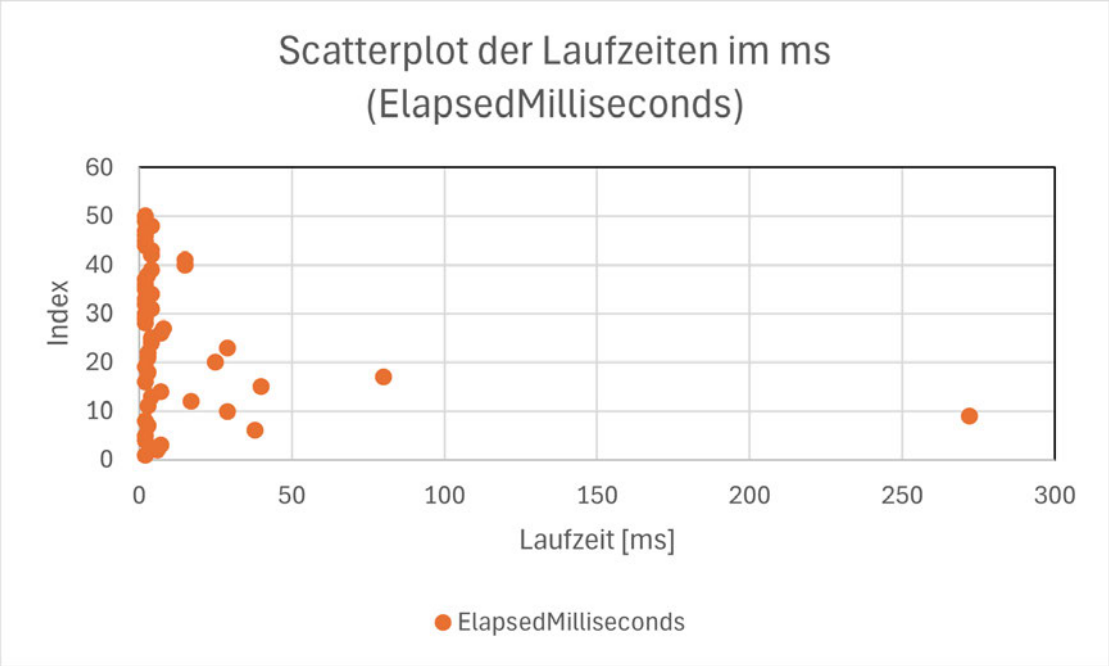

Abbildung 6.1: Werteverteilung der Laufzeit des Datensatzes einer `Overview`-API; Orange kennzeichnet die einzelnen Datenpunkte

Abbildung 6.1 zeigt eine eindeutige linkssteile Verteilung der Daten. Diese kommt zustande, da die meisten Aufrufe nur wenige Millisekunden für die Durchführung benötigen und nur eine geringe Anzahl über ca. 10 ms gehen. Die Methode der Standardabweichung kann nur bei einer Normalverteilung angewendet werden. Denn sie setzt voraus, dass die Daten symmetrisch um den Mittelwert verteilt sind und so anhand der Standardabweichung ein Vertrauensbereich bestimmt werden kann. Bei schiefen Verteilungen wird der Mittelwert von extremen Werten verschoben und dadurch die Standardabweichung überproportional groß. In Anbetracht dessen ist für die weitere Analyse der Interquartilsabstand implementiert worden.

6.3 IQR (Interquartilsabstand)

Die IQR-Methode fokussiert sich auf die Streuung der mittleren 50% der Daten. Der Interquartilsabstand wird berechnet als die Differenz zwischen dem 75(Q3) und 25(Q1) Perzentil. Ausreißer sind die Punkte, die unter dem 1,5-fachen des IQR unterhalb von Q1 und über dem 1,5-fachen oberhalb von Q3 [12]. Da diese Methode keine Normalverteilung der Daten voraussetzt, eignet sie sich für diesen Anwendungsfall. Die folgende Abbildung zeigt den Boxplot, welcher die Verteilung der Laufzeiten logarithmiert darstellt.

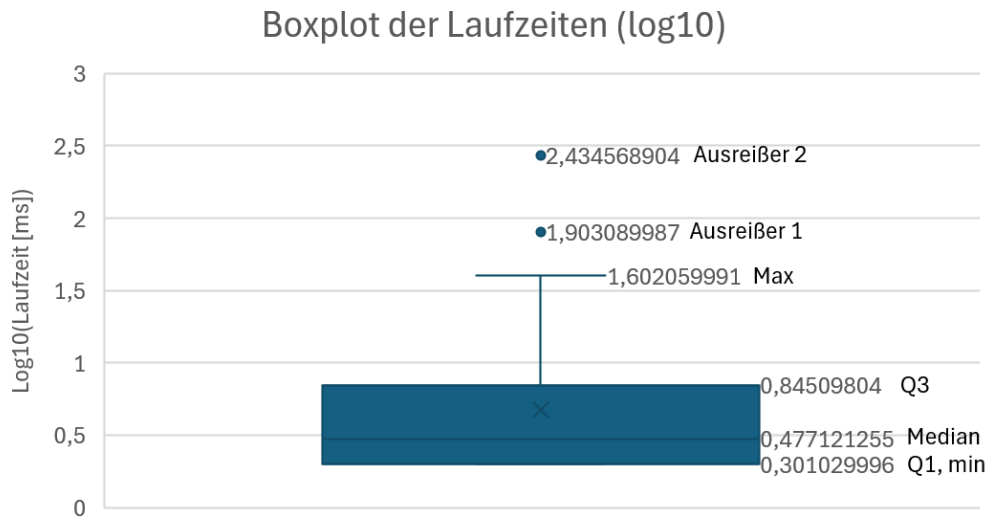

Abbildung 6.2: Boxplot der Laufzeit einer Overview-API; die y-Achse stellt die logarithmierte Laufzeit in [ms] dar

Die Abbildung 6.2 zeigt den Boxplot der gemessenen Laufzeiten, wobei die Laufzeit für eine bessere Übersicht logarithmisch skaliert ist. Es sind die Ausreißer, der Maximal- und Minimalwert (Whisker) und die Quartile zu erkennen. Die y-Achse ist logarithmisch skaliert, da die Ausreißer vergleichsweise hohe Werte haben und dadurch eine lineare Skalierung die Darstellung unübersichtlich macht. Umgerechnet ergeben die Kennwerte die folgenden Zeiten:

- $Q1, min \approx 2\,ms$
- $Median \approx 3\,ms$
- $Q3 \approx 7\,ms$
- $max \approx 40\,ms$
- $Ausreißer_1 \approx 80\,ms$
- $Ausreißer_2 \approx 272\,ms$

Das bedeutet, dass die mittleren 50% der gemessenen Laufzeiten im Bereich von 2 ms bis 7 ms liegen. Der geringe Median bei knapp 3 ms verdeutlicht ebenfalls, dass mehr als die Hälfte aller Messwerte sehr niedrig ausfallen.

Der obere Whisker reicht bis ungefähr 40 ms und definiert damit die obere Ausreißergrenze. Alle Werte, die über 40 ms liegen, können somit als Ausreißer klassifiziert werden (80 ms und 272 ms).

Der Boxplot zeigt, dass die `Overview`-API im Normalfall eine kurze Funktionsdauer hat, aber in Ausnahmefällen auch deutlich längere Laufzeiten auftreten können. Für die Log-Analyse sind die Ausreißer besonders wichtig, da sie Hinweise auf Probleme innerhalb des Systems liefern können.

6.4 Alternative Ausreißererkennung

Der Interquartilsabstand ist eine sehr einfache Methode, um Ausreißer zu identifizieren. Nach [38] gibt es neben der IQR-Methode drei weitere Erkennungsalgorithmen:

- **Z-Score**: Beschreibt, wie viele Standardabweichungen ein Datenpunkt vom Mittelwert entfernt ist. Eine häufig gesetzte Schwelle zur Klassifizierung eines Ausreißers sind ab einer Entfernung vom Mittelpunkt von 2,5 bis 3,5 Standardabweichungen.
- **DBSCAN**: Eine Clustering-Methode, welche dicht beieinanderliegende Datenpunkte clustert und entferntere Datenpunkte als Ausreißer markiert.
- **Isolation Forest**: Ein Verfahren, welches auf einer Baumstruktur basiert und das Ausreißer durch kurze Pfade (wenige Teilungen) erkennt. Beim „Isolation Forest“ muss definiert werden, wie viele Ausreißer erwartet sind. Basierend auf dieser Annahme wird ein Schwellenwert gesetzt, welcher die Datenpunkte als Ausreißer klassifiziert.

Eine sehr fortgeschrittene Variante für Loguntersuchungen wäre eine KI-gestützte Loganalyse, welche mit Datensätzen trainiert und dann zur Anomalieerkennung eingesetzt werden kann. Jedoch übersteigt dies den Rahmen dieser Arbeit. Da das Logging-System nur die Funktionsdauer als metrischen Wert besitzt und eine einfache Analysemethode genügt, wurde auf Effizienz gesetzt und eine Ausreißererkennung durch den Interquartilsabstand eingeführt.

7 Ergebnis

Dieses Kapitel stellt das Ergebnis des Auditing-Systems mit einer bekannten Schwachstelle dar.

7.1 Auditing-System

Das vollendete Logging-Modul für automatische Logs besteht aus drei Hauptteilen, die unabhängig voneinander für ihr eigenes Aufgabenspektrum verantwortlich sind und einer Mechanik zum Erstellen manueller Logs. Die Validierung zeigt, dass alle Anwendungsfälle zuverlässig abgedeckt werden. Die folgenden Anforderungen wurden erfüllt:

- **Interceptor:** APIs und Controller werden automatisch geloggt und neue Funktionen innerhalb dieser Klassen ebenfalls.
- **Event-Handler:** Exceptions werden protokolliert.
- **Action-Filter:** Eindeutige Trace-ID und die Argumente für einen Aufruf werden gesetzt und dem Event-Handler sowie der `AuditingHelper`-Klasse bereitgestellt.
- **Manuell:** Manuelle Log-Einträge können effizient erstellt werden und werden automatisch mit Kontextinformationen befüllt, sofern vorhanden.
- **Alle:** Protokollierte Funktionen besitzen eine eindeutige Trace-ID, welche eine Gruppierung mehrerer Einträge eines Aufrufs ermöglicht.
- **Alle:** Log-Daten werden einheitlichen in einem strukturierten Format dargestellt.
- **Alle:** Das Logging-Modul funktioniert unabhängig des System-Loggers (Log4Net)
- **Alle:** Das System ist unabhängig von Updates und vollständig modular. Jede zusätzlich benötigte Information kann einfach ergänzt werden.

Die folgende Abbildung stellt das gesamte Logging-System mit den jeweiligen Aufgabenbereichen dar:

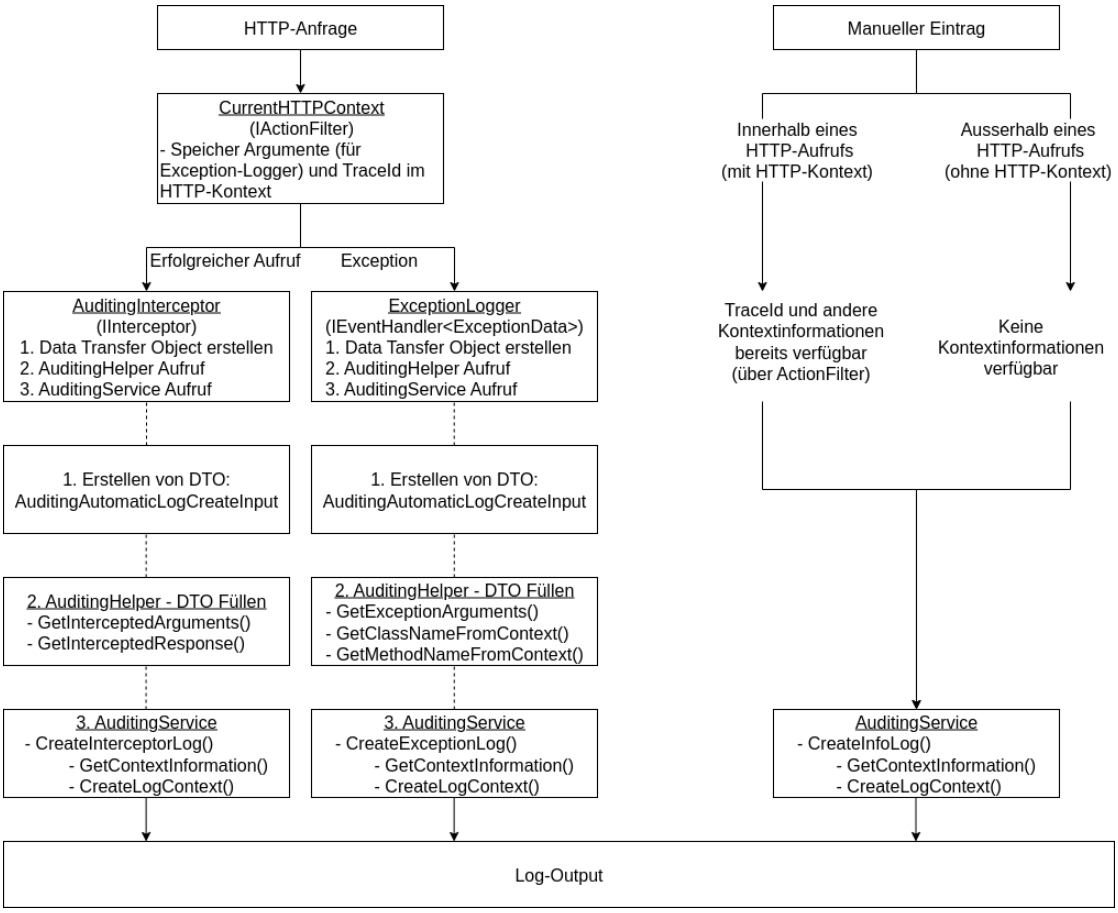

Abbildung 7.1: Architektur des Logging-Systems mit automatischer und manueller Protokollierung

Die Abbildung 7.1 gibt einen Einblick in die Architektur des Logging-Systems mit automatischer und manueller Protokollierung. Http-Anfragen der abgefangenen Klassen (`ApplicationService` und `PmControllerBase`) werden über den Interceptor und den Event-Handler automatisch erfasst, während manuelle Log-Einträge eine explizite Implementierung eines Entwicklers benötigen. Erfolgreiche Http-Anfragen werden über den `Interceptor` und Fehlerfälle vom `ExceptionLogger` protokolliert. Die vorgehensweise beider Verfahren ähnelt sich. Zuerst werden die DTOs für den Datentransfer an den `AuditingService` erstellt. Danach werden diese durch den `AuditingHelper` und die individuellen Objekte (beim Interceptor `IInvocation invocation` und

beim ExceptionLogger `ExceptionData eventData`) befüllt. Im Anschluss werden die DTOs an den `AuditingService` gereicht, welcher den Datensatz mit Kontextinformationen bereichert und den Log-Eintrag erstellt.

Für manuelle Einträge wird zwischen Logging innerhalb und außerhalb eines Http-Aufrufs unterschieden. Wird ein manueller Log-Eintrag innerhalb eines Http-Aufrufs erstellt, dann wurde zuvor eine API aufgerufen und die Trace-ID bereits durch den Action-Filter gesetzt. So werden bei der Darstellung der Log-Einträge (Abschnitt 5.2) jedem API-Aufruf die zugehörigen manuellen Einträge zugeordnet. Außerhalb eines Http-Aufrufs wird aufgrund fehlender Kontextinformationen der Log-Eintrag vom `AuditingService` direkt erstellt. Ein manueller Log-Eintrag ohne Kontextinformationen sollte jedoch nicht häufig auftreten, da dieses Modul entworfen wurde, um Nutzerinteraktionen nachzuverfolgen, welche wiederum einen Http-Aufruf erzeugen. Das System deckt alle Anwendungsfälle ab, jedoch ist die Datenqualität vom Aufrufkontext abhängig.

7.2 Bekannte Schwachstelle

Die Implementierung besitzt eine bekannte Schwachstelle: Sobald die Serilog-Konfiguration nicht mit der Struktur der zuvor erstellten Datenbanken und Tabellen übereinstimmt, schlägt das Speichern des gesamten Datensatzes fehl. Serilog benötigt die genauen Namen der Tabellenspalten (Tabelle 3.1) zum Schreiben von Einträgen. Stimmen diese nicht überein, versucht das Logging-Framework, Daten in eine nicht vorhandene Spalte zu schreiben und bricht die Aktion damit ab. Um dem entgegenzuwirken, wurde der sogenannte „Self-Log“ von Serilog eingebunden. Dadurch wird eine Konsolenausgabe erzeugt, sobald ein Fehler innerhalb des Logging-Frameworks selbst aufgetreten ist.

8 Ausblick

Das Auditing-System dient in erster Linie dazu, den technischen Hintergrund der Nutzeraktionen zu protokollieren. Log-Daten, die volle Transparenz über Aktionen bieten, schaffen nicht nur auf technischer Ebene eine Grundlage für verschiedenste Analysen, sondern öffnen auch die Möglichkeit Geschäftsprozesse zu unterstützen. Die gesammelten Daten können zukünftig in Kombination mit einem Management-Dashboard genutzt werden, welches beispielsweise Touren darstellt, die regelmäßig Probleme verursachen oder zu lange dauern.

Es ist bereits ein Widget in Planung, welches die `GetCommonExceptions`-API benutzt, um Fehlermeldungen mit ihren Fehlerquellen darzustellen. Der Benutzer soll die Möglichkeit bekommen, eine Zeitspanne und eine Exception auszuwählen. Basierend auf der Zeitspanne wird ein Dropdown mit den in diesem Zeitraum auftretenden Exceptions befüllt. Nach Auswahl einer Exception werden die verantwortlichen Quellen (Klassenname und Funktionsname) gelistet. Jede Quelle hat eine Häufigkeit, die anzeigt, wie oft die Exception von der jeweiligen Quelle ausgelöst wurde. Ein horizontaler Balken soll die verschiedenen Quellen basierend auf dem Anteil der Gesamtanzahl von Exception darstellen.

Ein ähnliches Widget kann mit der `GetMostUsedEndpoints`-API entwickelt werden, welches eine Übersicht über die häufigsten erfolgreichen Nutzeraktionen bietet.

9 Fazit

Die vorliegende Arbeit befasste sich mit dem Thema „Audit Logging und Änderungsanalyse in datengetriebenen Webanwendungen“ und zeigt, dass durch eine Kombination verschiedener Mechaniken erfolgreich ein unabhängiges System zur vollständigen Protokollierung von Nutzeraktionen in eine separate Datenbank implementiert werden kann.

Die Kernbausteine sind Interceptor, Action-Filter und Event-Handler. Die AuditingHelper-Klasse sorgt für eine einheitliche Darstellung der Daten und bietet Hilfsfunktionen, welche das Befüllen von Session-Informationen oder das Lesen aus dem Http-Kontext ermöglichen. Der Auditing-Service ruft die Helper-Klasse zum Befüllen der Data-Transfer-Objects auf und erstellt den Log-Eintrag. Die Log-Eigenschaften wurden anhand der „5 W's of Logging“ bestimmt.

Die Validierung belegt, dass erfolgreiche Aufrufe, Exceptions und manuelle Einträge automatisch mit den nötigen Kontextinformationen befüllt und anschließend korrekt protokolliert werden.

Mithilfe von APIs werden die Log-Information aus der Datenbank abgefragt und an das Frontend weitergegeben. Die Log-Einträge werden auf zwei verschiedenen Ebenen dargestellt. Zum einen der Übersicht und zum anderen der Detailansicht. Die Übersicht beinhaltet Filtermöglichkeiten und eine Tabelle mit Log-Daten, durch welche in die Detailansicht navigiert werden kann. Die Detailansicht bietet jede verfügbare Information zu einem Eintrag und stellt verwandte Einträge dar.

Die metrische Eigenschaft „ElapsedMilliseconds“, welche für die Laufzeit einer Funktion steht, wurde über den Interquartilsabstand untersucht, wodurch Ausreißer in der Laufzeit einer API-Funktion identifiziert werden konnten.

Ein Logging-System, welches volle Transparenz über Nutzeraktionen bietet, ist ein Grundstein für zahlreiche Analyse- und Darstellungsmöglichkeiten.

Literaturverzeichnis

[1] Ackermann, Philip: *Full Stack Web Development*. Rheinwerk Verlag, 2024. – ISBN 9781493224371

[2] AG, GFU C.: *Was bedeutet Angular*. – URL https://www.gfu.net/wiki/angular.html. – Zugriffsdatum: 19.08.2025

[3] Assecor: *C#*. 2025. – URL https://www.assecor.de/glossar/c-sharp. – Zugriffsdatum: 18.08.2025

[4] (AWS), Amazon Web S.: *Was ist JavaScript?*. – URL https://aws.amazon.com/de/what-is/javascript/. – Zugriffsdatum: 29.06.2025

[5] Boilerplate, ASP.NET: *Audit Logging*. – URL https://aspnetboilerplate.com/Pages/Documents/Audit-Logging. – Zugriffsdatum: 09.08.2025

[6] Carter, Phillip: *Introducing C# Source Generators*. 2020. – URL https://devblogs.microsoft.com/dotnet/introducing-c-source-generators/. – Zugriffsdatum: 10.08.2025

[7] contributors, MDN: *HTTP response status codes*. 2025. – URL https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status. – Zugriffsdatum: 12.09.2025

[8] contributors, MDN: *SPA (Single-page application)*. 2025. – URL https://developer.mozilla.org/en-US/docs/Glossary/SPA. – Zugriffsdatum: 18.10.2025

[9] Deinhard, Florian: *Was leistet das Entity Framework?* 2025. – URL https://www.it-schulungen.com/wir-ueber-uns/wissensblog/was-leistet-das-entity-framework.html. – Zugriffsdatum: 19.08.2025

[10] DOTNETFULLSTACKDEV: *Understanding Interceptors in .NET*. 2024. – URL https://dotnetfullstackdev.medium.com/understanding-interceptors-in-net-4574e2f67cab. – Zugriffsdatum: 24.07.2025

[11] GEEKSFORGEEKS: *Frontend vs Backend*. 2025. – URL https://www.geeksforgeeks.org/blogs/frontend-vs-backend/. – Zugriffsdatum: 18.07.2025

[12] GEEKSFORGEEKS: *Outliers and Outlier Detection*. 2025. – URL https://www.geeksforgeeks.org/data-analysis/what-is-outlier-detection/. – Zugriffsdatum: 08.09.2025

[13] GEEWAX, JJ: *API Design Patterns*. Manning, 2021. – ISBN 9781617295850

[14] GMBH fecher: *Framework: Definition & Wissenswertes*. 2025. – URL https://www.modernizing-applications.de/it-glossar/framework-definition/. – Zugriffsdatum: 18.08.2025

[15] GMBH impact.media: *Angular*. – URL https://www.impactmedia.de/wissen/angular. – Zugriffsdatum: 19.08.2025

[16] GOVINDARAJ, Aravind: *Differences Between Scoped, Transient, And Singleton Service*. 2022. – URL https://www.c-sharpcorner.com/article/differences-between-scoped-transient-and-singleton-service/. – Zugriffsdatum: 01.09.2025

[17] HARTZELL, Jimmy: *The Importance of Logging*. 2023. – URL https://www.thecodedmessage.com/posts/logging/. – Zugriffsdatum: 26.07.2025

[18] KALKAN, Halil: *Application Services*. 2024. – URL https://abp.io/docs/latest/framework/architecture/domain-driven-design/application-services. – Zugriffsdatum: 28.08.2025

[19] KINSTA: *Was ist TypeScript?* 2023. – URL https://kinsta.com/de/wissensdatenbank/was-ist-typescript/. – Zugriffsdatum: 25.06.2025

[20] KRYPCZYK, Veikko ; EZELL, Christopher: *Datenbanken*. entwickler.press, 2012. – ISBN 9783868024036

[21] LEARN, Microsoft: *EventHandler<TEventArgs> Delegate*. – URL https://learn.microsoft.com/en-us/dotnet/api/system.eventhandler-1?view=net-9.0. – Zugriffsdatum: 10.08.2025

[22] Learn, Microsoft: *Filters in ASP.NET Core*. 2024. – URL https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-9.0. – Zugriffsdatum: 10.08.2025

[23] Learn, Microsoft: *HttpContext in ASP.NET Core*. 2025. – URL https://learn.microsoft.com/en-us/aspnet/core/fundamentals/http-context?view=aspnetcore-9.0. – Zugriffsdatum: 17.07.2025

[24] Lewis, Alexander S.: *Objektorientierte Programmierung (OOP) – Definition*. 2024. – URL https://www.computerweekly.com/de/definition/Objektorientierte-Programmierung-OOP. – Zugriffsdatum: 18.07.2025

[25] Lock, Andrew: *ASP.NET Core in Action, Third Edition*. Manning Publications, 2023. – ISBN 9781638352587

[26] Mielebacher, Jörg: *Datenbanken für Nichtinformatiker*. Springer Vieweg, 2024. – ISBN 9783658426637

[27] Murugan, Mukesh: *When to Use Transient, Scoped, or Singleton in .NET Apps - Understanding Service Lifetimes*. 2025. – URL https://codewithmukesh.com/blog/when-to-use-transient-scoped-singleton-dotnet. – Zugriffsdatum: 01.09.2025

[28] Okta: *Data Transfer Object DTO Definition und Anwendungsbereich*. – URL https://www.okta.com/de-de/identity-101/dto/. – Zugriffsdatum: 26.08.2025

[29] Postman: *What is an API? A Beginner's Guide to APIs*. – URL https://www.postman.com/what-is-an-api/. – Zugriffsdatum: 25.07.2025

[30] Sagar, Gaurav: *Dependency Injection in C#: Understanding IoC Containers*. 2023. – URL https://medium.com/technology-nineleaps/dependency-injection-in-c-understanding-ioc-containers-72f0a303f681. – Zugriffsdatum: 23.08.2025

[31] Schmidt, Kevin J. ; Chuvakin, Anton A.: *Logging and Log Management*. 1. Syngress, 2012. – ISBN 9781597496360

[32] Sharif, Arfan: *Was ist strukturierte, unstrukturierte und halbstrukturierte Protokollierung?* 2023. – URL https://www.crowdstrike.com/de-de/cybersecurity-101/next-gen-siem/structured-logging/. – Zugriffsdatum: 21.08.2025

[33] Spiess-Informatik: *Was ist CSS?*. – URL https://www.spiess-informatik.de/was-ist-css/. – Zugriffsdatum: 25.06.2025

[34] Steinbrecher, Steffen: *C# für Dummies*. 1. Wiley-VCH, 2020. – ISBN 9783527715190

[35] Sullivan, William: *Javascript*. PublishDrive, 2017

[36] Talend: *Was ist eine API?*. – URL https://www.talend.com/de/resources/was-ist-eine-api/. – Zugriffsdatum: 25.07.2025

[37] Uzayr, Sufyan B.: *Frontend development*. URL https://doi.org/10.1201/9781003309062, 2022

[38] Widmann, Maarit: *Four Techniques for Outlier Detection*. 2021. – URL https://medium.com/low-code-for-advanced-data-science/four-techniques-for-outlier-detection-bf2346cbe077. – Zugriffsdatum: 09.09.2025

[39] Yelve, Vishal: *Explaning IActionFilter in ASP.NET Core*. 2024. – URL https://www.c-sharpcorner.com/article/explaning-iactionfilter-in-asp-net-core/. – Zugriffsdatum: 10.08.2025

A Anhang

A.1 Listings

A.1.1 HTML Beispiel

```
1  <!DOCTYPE html>
2  <html lang="de">
3  <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Einfache Webseite mit HTML</title>
7     <!-- CSS import für naechstes beispiel -->
8     <link rel="stylesheet" href="style.css">
9  </head>
10 <body>
11 <h1>Einfache Webseite mit HTML</h1>
12 <!-- Absatz -->
13 <p>Dies ist eine einfache HTML-Seite zur Demonstration. Sie zeigt, wie
       Inhalte strukturiert dargestellt werden können.</p>
14 <!-- Überschrift 2 -->
15 <h2>Inhalt</h2>
16 <!-- Ungeordnete Liste -->
17 <ul>
18    <li>Eine Liste</li>
19    <li>Ein Bild</li>
20    <li>Eine Verlinkung</li>
21 </ul>
22 <!-- Überschrift 3 -->
23 <h3>Beispielbild mit Inline-Formatierung</h3>
24 <img src="https://www.haw-hamburg.de/fileadmin/zentrale_PDF/
       Zentrale_Dokumente/Corporate_Design/HAW_Marke_CMYK_300dpi.jpg"
25     alt="Platzhalterbild"
26     style="width:25%;">
27 <!-- Link -->
```

```
28  <p>Verlinkung auf die HAW-Homepage <a href="https://www.haw-hamburg.de/
        bachelor-elektrotechnik-und-informationstechnik-studierende/" target=
        "_blank">HAW-Homepage</a>.</p>
29  <script src="script.js"></script>
30  </body>
31  <footer>
32     <p>2025 Peter Fischer</p>
33  </footer>
34  </html>
```

Listing A.1: HTML-Code der beispielhaften Seite

Im <head> wird zunächst der Kopf der Webseite definiert. Dieser Bereich dient dazu, dem Browser Metadaten[1], wie z. B. Zeichencodierung oder Titel des Tabs im Browser mitzuteilen. Inhalte dieses Tags sind auf der Webseite selbst nicht sichtbar, jedoch für die allgemeine Darstellung und die technische Funktionsweise notwendig.

Der Hauptbestandteil (Körper) einer HTML-Webseite ist das <body>-Element. Es enthält die Inhalte, die im Browser des Benutzers angezeigt werden sollen. In diesem Beispiel werden die erwähnten HTML-Elemente verwendet, um verschiedene Überschriften, eine Liste, ein Bild und einen Link zur HAW-Homepage zu erstellen.

Ein <footer> repräsentiert die Fußzeile. Typischerweise enthält dieser Informationen über den Autor, eine Verlinkung zum Impressum oder zu anderen Webseiten.

[1] Maschinenlesbare Informationen über das Dokument

A.1.2 CSS Beispiel

Der in Listing A.2 dargestellte Code zeigt die CSS-Formatierung der HTML-Seite.

```
1  /* Grundstil für den Seitenkörper */
2  body {
3      font-family: sans-serif;
4      padding: 20px;
5  }
6
7  /* Überschriften */
8  h1, h2, h3 {
9      color: #005293;
10 }
11
12 /* Ungeordnete Liste */
13 ul {
14     padding-left: 20px;
15 }
16
17 /* Bilder */
18 img {
19     width: 25%;
20     border: 1px solid;
21     border-radius: 5px;
22 }
23
24 /* Verlinkungen */
25 a {
26     color: #005293;
27     text-decoration: none;
28 }
```

Listing A.2: CSS-Code der beispielhaften Webseite

In Listing A.2 wird zunächst im Seitenkörper die Schriftart geändert und ein Rand von 20 Pixeln definiert. Die Überschriften werden eingefärbt und die ungeordnete Liste erhält einen weiteren Abstand von 20 Pixeln zum linken Rand. Das Logo der HAW-Hamburg wird auf eine Größe von 25% skaliert, mit einem Rand versehen und einer Rundung für die Kanten mit einem Radius von 5 Pixeln. Zum Schluss wird die Verlinkung eingefärbt und jegliche Dekoration (Unterstrich bei Hyperlinks) entfernt.

A.1.3 SQL Beispiel

```sql
-- Data Definition Language:
-- Datenbank erstellen
CREATE DATABASE IF NOT EXISTS thesis;

-- Datenbank auswaehlen
USE thesis;

-- Tabelle erstellen
CREATE TABLE teilnehmer (
   personen_id INT PRIMARY KEY AUTO_INCREMENT,
   vorname VARCHAR(50),
   nachname VARCHAR(50)
);

-- Data Manipulation Language:
-- Daten hinzufuegen
INSERT INTO teilnehmer (vorname, nachname) VALUES
('Henning', 'Dierks'),
('Ulrike', 'Herster'),
('Peter', 'Fischer');

-- Data Query Language:
-- Daten abfragen (Teilnehmer mit dem Namen Peter)
SELECT vorname, nachname
FROM teilnehmer
WHERE vorname = 'Peter';
```

Listing A.3: SQL Skript zum Demonstrieren der verschiedenen SQL-Kategorien

Dieses Skript dient als Beispiel für den Einsatz der drei SQL-Kategorien DDL, DML und DQL. Es zeigt, wie mit DDL eine Datenbank mit Tabelle erstellt, durch DML mit Daten befüllt und anschließend mittels DQL abgefragt wird. Das Resultat der Abfrage liefert den Datensatz:

vorname	nachname
Peter	Fischer

A.2 Relevanter Code des Auditing-Systems

A.2.1 Entität: AuditLog

```
public class AuditLog : Entity<long>
{
    public DateTime TimeStamp { get; set; }
    public LogLevel LogLevel { get; set; }
    public DateTime? StartTime { get; set; }
    public DateTime? EndTime { get; set; }
    public long? ElapsedMilliseconds { get; set; }
    public bool IsManual { get; set; }
    public int? StatusCode { get; set; }
    public string Message { get; set; }
    public string Details { get; set; }
    public string ExceptionType { get; set; }
    public string Exception { get; set; }
    public string InnerException { get; set; }
    public int? TenantId { get; set; }
    public long? UserId { get; set; }
    public string UserName { get; set; }
    public string ClassName { get; set; }
    public string MethodName { get; set; }
    public string Arguments { get; set; }
    public string Response { get; set; }
    public Guid? TraceId { get; set; }
    public AuditLog()
    {
        TimeStamp = DateTime.Now;
    }
}
```

Listing A.4: Entität, welche die Tabelle in der Datenbank widerspiegelt; jede Eigenschaft inkl. Datentyp stellt eine Spalte in der Tabelle dar

Anhand dieser Entität wird die Tabelle „AuditLogs“ innerhalb der Datenbank „PmAuditLogDb“ erstellt. Die `AuditLog`-Entität bildet damit die Grundlage für das Speichern und die spätere Auswertung der Log-Einträge. In Kombination mit dem `AuditingService` und dem `AuditingHelper` entsteht ein geschlossenes System, das sowohl manuelle als auch automatische Log-Einträge erfasst und in der Datenbank ablegt.

A.2.2 Serilog-Konfiguration

```
"Serilog": {
  "Using": [ "Serilog.Sinks.MSSqlServer" ],
  "MinimumLevel": {
    "Default": "Information",
    "Override": {
      "Microsoft": "Fatal",
      "System": "Fatal",
      "Quartz": "Fatal"
    }
  },
  "WriteTo": [
    {
      "Name": "MSSqlServer",
      "Args": {
        "connectionString": "PmAuditLogDb",
        "tableName": "AuditLogs",
        "autoCreateSqlTable": false,
        "columnOptionsSection": {
          "removeStandardColumns": [ "Properties", "MessageTemplate", "
              LogEvent", "Level" ],
          "additionalColumns": [
            {
              "ColumnName": "LogLevel",
              "DataType": "int"
            },
            {
              "ColumnName": "StatusCode",
              "DataType": "int"
            },
            {
              "ColumnName": "TraceId",
              "DataType": "uniqueidentifier"
            },
            {
              "ColumnName": "IsManual",
              "DataType": "bit"
            },
            {
              "ColumnName": "ExceptionType",
```

```
      "DataType": "nvarchar"
    },
    {
      "ColumnName": "UserName",
      "DataType": "nvarchar"
    },
    {
      "ColumnName": "Details",
      "DataType": "nvarchar"
    },
    {
      "ColumnName": "InnerException",
      "DataType": "nvarchar"
    },
    {
      "ColumnName": "Response",
      "DataType": "nvarchar"
    },
    {
      "ColumnName": "UserId",
      "DataType": "bigint"
    },
    {
      "ColumnName": "TenantId",
      "DataType": "int"
    },
    {
      "ColumnName": "StartTime",
      "DataType": "datetime"
    },
    {
      "ColumnName": "EndTime",
      "DataType": "datetime"
    },
    {
      "ColumnName": "ElapsedMilliseconds",
      "DataType": "bigint"
    },
    {
      "ColumnName": "ClassName",
```

```
            "DataType": "nvarchar",
          },
          {
            "ColumnName": "MethodName",
            "DataType": "nvarchar",
          },
          {
            "ColumnName": "Arguments",
            "DataType": "nvarchar",
            "DataLength": -1
          }
        ]
      },
      "columnOptions": {
        "store": [
          "Message",
          "TimeStamp",
          "Exception"
        ]
      }
    }
  }
 ],
},
```

Listing A.5: Konfiguration des Logging-Frameworks Serilog

A.2.3 Log-Level Enum

```
1 public enum LogLevel
2 {
3    All = 0,
4    Information = 1,
5    Debug = 2,
6    Warning = 4,
7    Error = 8,
8 }
```

Listing A.6: Enum zum Definieren der Wichtigkeit eines Log-Eintrags

Das Enum `LogLevel` definiert die Wichtigkeit eines Eintrags. Dadurch wird sichergestellt, dass Log-Einträge einheitlich klassifiziert und konsistent in der Datenbank abgelegt werden können. Dieses Enum wird ebenfalls im Frontend zum Filtern nach Log-Einträgen verwendet. Der Eintrag „Alle“ ist vorhanden, um in einem Filter alle Log-Einträge anzuzeigen.

A.2.4 Interceptor

```
1  // Interceptor, der Methodenaufrufe unterbricht und automatisch Logs durch
       den AuditingService erzeugt.
2  public class AuditingInterceptor : IInterceptor
3  {
4     // Service, der die Log-Eintraege erstellt
5     private readonly AuditingService _auditingService;
6     // Standard Log-Nachricht
7     private const string Message = "Completed";
8     // Methoden, die nicht geloggt werden sollen
9     private static readonly List<string> LoggingBlacklist = new List<string>
10    {
11       "IncludeAll",
12       "FilterNotPaginated",
13       "GetAll",
14       "OnActionExecuting",
15       "OnActionExecuted",
16       "OnActionExecutionAsync"
17    };
18    // Konstruktor bekommt ueber DI (Dependency-Injection) den AuditingService
          Injiziert
19    public AuditingInterceptor(AuditingService auditingService)
20    {
21       _auditingService = auditingService;
22    }
23    // Kernfunktion des Interceptors. Fuerht die aufgerufene Methode aus,
          misst die Laufzeit und erstellt anschliessend einen Log-Eintrag
24    public void Intercept(IInvocation invocation)
25    {
26       string methodName = invocation.Method?.Name ?? "Unknown";
27       // Falls die aufgerufene Methode in der Blacklist vorkommt, ohne
             logging Fortfahren
28       if (LoggingBlacklist.Contains(methodName))
29       {
30          invocation.Proceed();
```

```
31            return;
32        }
33        string className = invocation.TargetType?.Name ?? "Unknown";
34        // Startzeit der Funktion auf den jetztigen Zeitpunkt setzen
35        DateTime startTime = DateTime.Now;
36        // Stoppuhr starten
37        Stopwatch stopwatch = Stopwatch.StartNew();
38        //Fortfahren mit der unterbrochenen Funktion bis zum Abschluss
39        invocation.Proceed();
40        // Stoppuhr stoppen
41        stopwatch.Stop();
42        // Endzeit aus Startzeit und Dauer berechnen
43        DateTime endTime = startTime.AddMilliseconds(stopwatch.
              ElapsedMilliseconds);
44        // DTO, zum strukturieren der Log-Daten
45        AuditingAutomaticLogCreateInput createInput = new
              AuditingAutomaticLogCreateInput()
46        {
47            LogLevel = LogLevel.Information,
48            Message = Message,
49            StartTime = startTime,
50            EndTime = endTime,
51            ElapsedMilliseconds = stopwatch.ElapsedMilliseconds,
52            MethodName = methodName,
53            ClassName = className,
54            Arguments = AuditingHelper.GetInterceptedArguments(invocation),
55            Response = AuditingHelper.GetInterceptedResponse(invocation)
56        };
57        // Service-Funktion, welche als parameter ein
              AuditingAutomaticLogCreateInput-Objekt uebergibt und einen Log-
              Eintrag erstellt.
58        _auditingService.CreateInterceptorLog(createInput);
59    }
60 }
```

Listing A.7: Implementierung des AuditingIntercerptors

Der `AuditingInterceptor` bietet die Grundlage für das automatische Protokollieren von bestimmten Methodenaufrufen. Er ermöglicht es, Methodenaufrufe abzufangen, bevor sie ausgeführt werden können. Auf diese Weise wird das Loggen ohne Änderungen am ursprünglichen Code möglich. Damit der Interceptor die gewünschten Klassen und Interfaces unterbrechen kann, wird er dafür innerhalb der `AuditingInterceptorContributor`-Klasse registriert.

A.2.5 Registrierung: AuditingInterceptorContributor

```
1  // Registriert den Interceptor automatisch fuer bestimmte Komponenten (
       ApplicationServices und Controller)
2  public class AuditingInterceptorContributor :
       IContributeComponentModelConstruction
3  {
4     // Wird von Castle Windsor beim Erstellen des ComponentModels aufgerufen.
5     public void ProcessModel(IKernel kernel, ComponentModel model)
6     {
7        // Interceptor hinzufuegen, falls die Komponente ein
             IApplicationService ist
8        if (typeof(IApplicationService).IsAssignableFrom(model.Implementation))
9        {
10          model.Interceptors.Add(new InterceptorReference(typeof(
                AuditingInterceptor)));
11       }
12
13       // Interceptor hinzufuegen, falls die Komponente eine PmControllerBase
             ist
14       if (typeof(PmControllerBase).IsAssignableFrom(model.Implementation))
15       {
16          model.Interceptors.Add(new InterceptorReference(typeof(
                AuditingInterceptor)));
17       }
18    }
19 }
```

Listing A.8: Implementierung des AuditingInterceptorContributor

Listing A.8 zeigt, wie der Interceptor für die Klasse `PmControllerBase` und das Interface `IApplicationService` registriert wird.

Anschließend wird der `AuditingInterceptorContributor` im IoC-Container eingebunden, sodass Castle Windsor (Inversion of Control Container) beim Aufbau der Komponenten die Contributor-Klasse berücksichtigt und die ausgewählten Komponenten (Application Services und Controller) mit dem Auditing-Interceptor erweitert. Der folgende Codeausschnitt zeigt die Registrierung des `AuditingInterceptorContributor` im IoC-Container.

```
Configuration.IocManager.IocContainer.Kernel.ComponentModelBuilder
    .AddContributor(new AuditingInterceptorContributor());
```

Listing A.9: Registrierung des AuditingInterceptorContributor im IoC-Container

A.2.6 Action-Filter

```
// Implementierung eines ActionFilters, der es ermoeglicht, Daten im
    HttpContext abzulegen.
public class CurrentHttpContext : IActionFilter
{
    // Funktion, die vor durchfuehrung der HTTP-Anfrage ausgefuehrt wird
    public void OnActionExecuting(ActionExecutingContext context)
    {
        // Argumente werden im HttpContext gespeichert
        context.HttpContext.Items["Arguments"] = context.ActionArguments;
        // Neue TraceId wird als GUID fuer diese Action erstellt
        context.HttpContext.Items["TraceId"] = Guid.NewGuid();
    }
    // Funktion, die vom Interface gefordert wird, jedoch keine verwendung hat
    public void OnActionExecuted(ActionExecutedContext context)
    {
    }
}
```

Listing A.10: ActionFilter CurrentHttpContext

In der eigens erstellten Klasse `CurrentHttpContext` werden für das Logging in dem Http-Kontext zwei Eigenschaften gespeichert. Zum einen die vom Client an den Server übergebenen Argumente und zum anderen wird für jeden Kontext eine Trace-ID erstellt und gespeichert. Diese ID wird verwendet, um mehrere Log-Einträge, die in derselben Funktion entstehen, sei es z. B. durch einen automatischen Eintrag des Interceptors und einen weiteren manuellen Eintrag, zu unterscheiden.

A.2.7 Event-Handler

```
// Event-Handler fuer ABP-Exception-Events. Reagiert auf ExceptionData und
    schreibt einen Log-Eintrag durch den AuditingService.
public class ExceptionLogger : IEventHandler<ExceptionData>
{
   private readonly AuditingService _auditingService;
   private readonly AuditingHelper _auditingHelper;
   // Dependency Injection
   public ExceptionLogger(AuditingService auditingService, AuditingHelper
       auditingHelper)
   {
      _auditingService = auditingService;
      _auditingHelper = auditingHelper;
   }
   // Wird bei einem Exception-Event vom Event-Bus aufgerufen
   public void HandleEvent(ExceptionData eventData)
   {
     // Data-Transfer-Object zum speichern der Exception-Daten als ein Objekt
      AuditingAutomaticLogCreateInput createInput = new
          AuditingAutomaticLogCreateInput()
      {
         Exception = eventData.Exception,
         Arguments = _auditingHelper.GetExceptionArguments(),
         MethodName = _auditingHelper.GetMethodNameFromContext(),
         ClassName = _auditingHelper.GetClassNameFromContext()
      };
      /*
        Auswahl von verschiedenen ExceptionTypen.
        Da das Framework weiss, welche Exception genau geworfen wird,
        wird ein automatischer Downcast von Exception auf die jeweilige
        Unterklasse durchgefuehrt.
      */
      switch (createInput.Exception)
      {
         case AbpValidationException validationEx:
            createInput.Details = string.Join("; ",
               validationEx.ValidationErrors.Select(e => e.ErrorMessage));
            break;
         case UserFriendlyException userFriendlyException:
            createInput.Details = userFriendlyException.Details;
            break;
      }
      // Uebergabe des Auditing-Objekts an den Auditingservice
```

```
40          _auditingService.CreateExceptionLog(createInput);
41      }
42  }
```

Listing A.11: ExceptionLogger EventHandler

Der `ExceptionLogger` implementiert das `IEventHandler<ExceptionData>`-Interface und reagiert somit auf ausgelöste `ExceptionData`-Events des ASP.NET Boilerplate-Frameworks.

A.2.8 AuditingService

```
1  // Service zum Erstellen von Log-Eintraegen
2  public class AuditingService: ISingletonDependency
3  {
4      private readonly AuditingHelper _auditingHelper;
5
6      // Dependency Injection
7      public AuditingService(AuditingHelper auditingHelper)
8      {
9          _auditingHelper = auditingHelper;
10     }
11     // Manuellen Info-Log schreiben
12     public void CreateInfoLog(
13         string message,
14         object arguments = null,
15         [CallerMemberName] string callerMethod = "",
16         [CallerFilePath] string callerPath = "")
17     {
18         CreateLog(LogLevel.Information, message, arguments, callerMethod,
               callerPath);
19     }
20
21     // Manuellen Debug-Log schreiben
22     public void CreateDebugLog(
23         string message,
24         object arguments = null,
25         [CallerMemberName] string callerMethod = "",
26         [CallerFilePath] string callerPath = "")
27     {
28         CreateLog(LogLevel.Debug, message, arguments, callerMethod, callerPath)
               ;
```

```
29      }
30
31      // Manuellen Warn-Log schreiben
32      public void CreateWarnLog(
33          string message,
34          object arguments = null,
35          [CallerMemberName] string callerMethod = "",
36          [CallerFilePath] string callerPath = "")
37      {
38          CreateLog(LogLevel.Warning, message, arguments, callerMethod,
                callerPath);
39      }
40
41      // Manuellen Error-Log schreiben
42      public void CreateErrorLog(
43          string message,
44          object arguments = null,
45          [CallerMemberName] string callerMethod = "",
46          [CallerFilePath] string callerPath = "")
47      {
48          CreateLog(LogLevel.Error, message, arguments, callerMethod, callerPath)
                ;
49      }
50
51      // Gemeinsame Log-Methode fuer alle manuellen Logs
52      private void CreateLog(
53          LogLevel level,
54          string message,
55          object arguments = null, // optional
56          string callerMethod = "",
57          string callerPath = "")
58      {
59          // Entscheide anhand des Parameters level, welche Serilog-Funktion (Log
                .Error, Log.Information, ..) ausgefuerht werden soll
60          Action<Exception, string> logAction = AuditingHelper.GetLogAction(level
                );
61          // Befuelle DTO fuer manuelle Logs
62          LogContextInformation li = new LogContextInformation()
63          {
64              IsManual = true,
65              LogLevel = level,
66              ContextInformation = _auditingHelper.GetContextInformation(
                    callerMethod, callerPath),
67              Arguments = AuditingHelper.SerializeOrNull(arguments),
```

```
68          Message = message
69      };
70      // Erstellt den Serilog Log-Kontext und ruft dort die gewaehlte Serilog
            -Funktion aus
71      CreateLogContext(li, logAction);
72  }
73
74  // Log-Methode fuer automatische Logs durch den Interceptor
75  internal void CreateInterceptorLog(AuditingAutomaticLogCreateInput input)
76  {
77      Action<Exception, string> logAction = AuditingHelper.GetLogAction(input
            .LogLevel);
78
79      // Befuelle DTO fuer automatische Interceptor Logs
80      LogContextInformation li = new LogContextInformation()
81      {
82          IsManual = false,
83          LogLevel = input.LogLevel,
84          ContextInformation = _auditingHelper.GetContextInformation(input.
                MethodName, input.ClassName),
85          Arguments = AuditingHelper.SerializeArgument(input.Arguments),
86          Response = AuditingHelper.SerializeOrNull(input.Response),
87          ElapsedMilliseconds = input.ElapsedMilliseconds,
88          Message = input.Message,
89          StartTime = input.StartTime,
90          EndTime = input.EndTime,
91      };
92      // Erstellt den Serilog Log-Kontext und ruft dort die gewaehlte Serilog
            -Funktion aus
93      CreateLogContext(li, logAction);
94  }
95
96  // Log-Methode fuer automatische Logs durch den ExceptionLogger (
        EventHandler)
97  internal void CreateExceptionLog (AuditingAutomaticLogCreateInput input)
98  {
99      Action<Exception, string> logAction = Log.Error; // Bei einem Fehler
            immer die Serilog-Funktion Log.Error ausfuehren
100     // Befuelle DTO fuer automatische Exception Logs
101     LogContextInformation li = new LogContextInformation()
102     {
103         IsManual = false,
104         LogLevel = LogLevel.Error,
105         Exception = input.Exception,
```

```
106             ExceptionType = input.Exception?.GetType().Name,
107             InnerException = input.Exception?.InnerException?.Message,
108             ContextInformation = _auditingHelper.GetContextInformation(input.
                    MethodName, input.ClassName),
109             Arguments = AuditingHelper.SerializeArgument(input.Arguments),
110             Message = input.Exception?.Message,
111             Details = input.Details
112         };
113
114         // Erstellt den Serilog Log-Kontext und ruft dort Log.Error auf
115         CreateLogContext(li, logAction);
116     }
117
118     // Baut LogContext und fuehrt Logging aus
119     private static void CreateLogContext(LogContextInformation logInformation,
            Action<Exception, string> logAction)
120     {
121         // LogContext wird anhand der uebergebenen Parameter aufgebaut
122         using(LogContext.PushProperty("LogLevel", (int)logInformation.LogLevel)
                )
123         using(LogContext.PushProperty("IsManual", logInformation.IsManual))
124         using(LogContext.PushProperty("Message", logInformation.Message))
125
126         using(LogContext.PushProperty("StartTime", logInformation.StartTime))
127         using(LogContext.PushProperty("EndTime", logInformation.EndTime))
128         using(LogContext.PushProperty("ElapsedMilliseconds", logInformation.
                ElapsedMilliseconds))
129
130         using(LogContext.PushProperty("ClassName", logInformation.
                ContextInformation.ClassName))
131         using(LogContext.PushProperty("MethodName", logInformation.
                ContextInformation.MethodName))
132         using(LogContext.PushProperty("TenantId", logInformation.
                ContextInformation.MandantId))
133         using(LogContext.PushProperty("UserId", logInformation.
                ContextInformation.UserId))
134         using(LogContext.PushProperty("UserName", logInformation.
                ContextInformation.UserName))
135         using(LogContext.PushProperty("TraceId", logInformation.
                ContextInformation.TraceId))
136         using(LogContext.PushProperty("StatusCode", logInformation.
                ContextInformation.StatusCode))
137
138         using(LogContext.PushProperty("Exception", logInformation.Exception))
```

```
139         using(LogContext.PushProperty("Details", logInformation.Details))
140         using(LogContext.PushProperty("ExceptionType", logInformation.
                ExceptionType))
141         using(LogContext.PushProperty("InnerException", logInformation.
                InnerException))
142
143         using(LogContext.PushProperty("Arguments", logInformation.Arguments))
144         using(LogContext.PushProperty("Response", logInformation.Response))
145         {
146             // tatsaechlicher Log-Aufruf
147             logAction(logInformation.Exception, logInformation.Message);
148         }
149     }
150     // Data-Transfer-Object fuer Kontext Informationen
151     private class LogContextInformation
152     {
153         public LogLevel LogLevel { get; set; }
154         public bool IsManual { get; set; }
155         public string Message { get; set; }
156         public DateTime? StartTime { get; set; }
157         public DateTime? EndTime { get; set; }
158         public long? ElapsedMilliseconds { get; set; }
159         public ContextInformation ContextInformation { get; set; }
160
161         [CanBeNull] public Exception Exception { get; set; }
162         [CanBeNull] public string Details { get; set; }
163         public string ExceptionType { get; set; }
164         public string InnerException { get; set; }
165         public string Arguments { get; set; }
166         public string Response { get; set; }
167     }
168 }
```

Listing A.12: AuditingService-Klasse zum erstellen der Log-Einträge

Für jeden Log-Eintrag wird ein Data-Transfer-Object `LogContextInformation` erzeugt, das alle Eigenschaften der Haupt-Entität `AuditLog` (Listing A.4) enthält. Durch `LogContext.PushProperty(SSpaltenname", Variable)` werden Variablen den Tabellenspalten zugeordnet und über den MSSql-Sink in die Datenbank geschrieben.

Die Attribute `[CallerMemberName]` und `[CallerFilePath]` werden lediglich für die manuellen Log-Funktionen verwendet. Akzeptiert eine Funktion diese Attribute als

Parameter, so erhält sie den Namen der aufrufenden Methode und den vollständigen Dateipfad der aufrufenden Datei über das .NET-Framework.

Der Methodenname kann so übernommen werden, wie die Funktion ihn übergeben bekommt, der Dateiname muss aus dem gesamten Dateipfad extrahiert werden. Dazu gibt es bereits eine Funktion des Frameworks `GetFileNameWithoutExtension()`

A.2.9 ContextInformation

```
1  public class ContextInformation
2  {
3     public string MethodName { get; set; }
4     public string ClassName { get; set; }
5     public int? TenantId { get; set; }
6     public long? UserId { get; set; }
7     public string UserName { get; set; }
8     public int? StatusCode { get; set; }
9     public Guid? TraceId { get; set; }
10 }
```

Listing A.13: ContextInformation DTO

Diese Klasse wurde erstellt, um die Lesbarkeit innerhalb des `LogContextInformation`-DTO zu verbessern. Ebenfalls kann dadurch in der `GetContextInformation(string methodName, string sourceFilePath)`-Funktion der `AuditingHelper`-Klasse ein `ContextInformation`-Objekt zurückgegeben werden, um alle Eigenschaften nur einem Funktionsaufruf zu setzen (Listing A.12 Z. 123).

A.2.10 AuditingHelper

```
// Hilfsklasse fuer Audit-Logs
public class AuditingHelper : ITransientDependency
{
    private const int ClassSegmentIndex = 3;
    private const int ClassSegmentNumber = ClassSegmentIndex + 1;

    private const int MethodSegmentIndex = 4;
    private const int MethodSegmentNumber = MethodSegmentIndex + 1;

    private readonly IAbpSession _abpSession;
    private readonly IHttpContextAccessor _httpContextAccessor;
    public AuditingHelper(IAbpSession abpSession, IHttpContextAccessor
        httpContextAccessor)
    {
        _abpSession = abpSession;
        _httpContextAccessor = httpContextAccessor;
    }

    // Holt den Klassennamen aus dem Request-Pfad fuer den ExceptionLogger
    public string GetClassNameFromContext()
    {
        string path = _httpContextAccessor.HttpContext?.Request?.Path.Value;
        if (string.IsNullOrEmpty(path))
            return null;
        string[] segments = SplitSegments(path);
        return segments.Length < ClassSegmentNumber ? null : $"{segments[
            ClassSegmentIndex]}AppService";
    }

    // Holt den Methodennamen aus dem Request-Pfad fuer den ExceptionLogger
    public string GetMethodNameFromContext()
    {
        string path = _httpContextAccessor.HttpContext?.Request?.Path.Value;
        if (string.IsNullOrEmpty(path))
            return null;
        string[] segments = SplitSegments(path);
        return segments.Length < MethodSegmentNumber ? null : segments[
            MethodSegmentIndex];
    }

    // Holt aus dem IInvocation-Objekt die Parameter, die bei einem
        Methodenaufruf an den Interceptor uebergeben wurden
```

```
39    public static List<AuditingLogArgument> GetInterceptedArguments(
          IInvocation invocation)
40    {
41       if (invocation.Arguments.Length == 0)
42          return null;
43
44       var argumentList = new List<AuditingLogArgument>();
45       for (int i = 0; i < invocation.Arguments.Length; i++)
46       {
47          argumentList.Add(new AuditingLogArgument()
48          {
49             Name = invocation.Method.GetParameters()[i].Name,
50             Value = invocation.Arguments[i]
51          });
52       }
53       return argumentList;
54    }
55
56    // Holt den Rueckgabewert einer Methode die vom Interceptor abgefangen
          wurde.
57    public static object GetInterceptedResponse(IInvocation invocation)
58    {
59       if (invocation.ReturnValue == null)
60          return null;
61
62       Type returnType = invocation.ReturnValue.GetType();
63       if (returnType.IsGenericType && returnType.GetProperty("Result") !=
             null)
64          return returnType.GetProperty("Result")?.GetValue(invocation.
                ReturnValue);
65
66       return invocation.ReturnValue;
67    }
68
69    // Holt Parameter aus HttpContext.Items fuer den ExceptionLogger und
          formatiert diese
70    public List<AuditingLogArgument> GetExceptionArguments()
71    {
72       if (_httpContextAccessor.HttpContext?.Items["Arguments"] == null)
73          return null;
74
75       var arguments = new List<AuditingLogArgument>();
76       if (_httpContextAccessor.HttpContext?.Items["Arguments"] is IDictionary
             <string, object> methodArgs)
```

```
77          {
78              foreach (var entry in methodArgs)
79              {
80                  arguments.Add(new AuditingLogArgument()
81                  {
82                      Name = entry.Key,
83                      Value = entry.Value
84                  });
85              }
86          }
87          return arguments.Count == 0 ? null : arguments;
88      }
89
90      // Serialisiert Argumente in einen Json-String
91      public static string SerializeArgument(List<AuditingLogArgument> arguments
            )
92      {
93          if (arguments == null || arguments.Count == 0)
94              return null;
95          var sb = new StringBuilder();
96          foreach (var entry in arguments)
97          {
98              sb.Append(SerializeOrNull(new Dictionary<string, object> { { entry.
                    Name, entry.Value } }));
99          }
100         return sb.ToString();
101     }
102
103     // Serialisiert ein Objekt
104     public static string SerializeOrNull(object obj)
105     {
106         if (obj == null)
107             return null;
108
109         try
110         {
111             // Ist das Objekt eine Auflistung aber kein String
112             if (obj is IEnumerable enumerable && !(obj is string))
113             {
114                 var enumerator = enumerable.GetEnumerator(); // Zum iterieren
                        durch die Elemente
115                 bool isEmpty = !enumerator.MoveNext(); // Pruefen, ob inhalte
                        vorhanden sind
```

```
116             (enumerator as IDisposable)?.Dispose(); // Ressourcenfreigabe,
                    wenn moeglich
117             if (isEmpty) {
118                return null;
119             }
120          }
121          string serialized = JsonConvert.SerializeObject(obj);
122          if (serialized == "[]" || serialized == "{}" || string.
                 IsNullOrWhiteSpace(serialized))
123             return null;
124
125          return serialized;
126       }
127       catch
128       {
129          return null;
130       }
131    }
132
133    // Baut Kontextinformationen fuer einen Audit-Log zusammen
134    public ContextInformation GetContextInformation(string methodName, string
          sourceFilePath)
135    {
136       string className = Path.GetFileNameWithoutExtension(sourceFilePath);
137       return new ContextInformation
138       {
139          ClassName = className,
140          MethodName = methodName,
141          UserName = GetUserNameFromContext(),
142          TraceId = GetTraceIdFromContext(),
143          MandantId = _abpSession.TenantId,
144          UserId = _abpSession.UserId,
145          StatusCode = GetStatusCodeFromContext()
146       };
147    }
148
149    // Gibt die passende Log-Action abhaengig vom Log-Level zurueck
150    public static Action<Exception, string> GetLogAction(LogLevel level)
151    {
152       switch (level)
153       {
154          case LogLevel.Warning: return Log.Warning;
155          case LogLevel.Error: return Log.Error;
156          case LogLevel.Information:
```

```
157             case LogLevel.Debug:
158             default: return Log.Information;
159         }
160     }
161
162     // Teilt einen Pfad in Segmente
163     private static string[] SplitSegments(string path)
164     {
165         return path.Split(new[] { '/' }, StringSplitOptions.RemoveEmptyEntries)
                ;
166     }
167
168     // Holt den Benutzernamen aus dem HttpContext
169     private string GetUserNameFromContext()
170     {
171         return _httpContextAccessor.HttpContext?.User?.Identity?.Name;
172     }
173
174     // Holt eine TraceId falls vorhanden
175     private Guid? GetTraceIdFromContext()
176     {
177         if (_httpContextAccessor.HttpContext?.Items["TraceId"] is Guid traceId)
178             return traceId;
179         return null;
180     }
181
182     // Holt den StatusCode aus der Response
183     private int? GetStatusCodeFromContext()
184     {
185         HttpContext context = _httpContextAccessor.HttpContext;
186         return context?.Response?.StatusCode ?? StatusCodes.
                Status401Unauthorized;
187     }
188 }
```

Listing A.14: Hilfsklasse AuditingHelper für übergreifende Logik

Die `AuditingHelper`-Klasse wurde entwickelt, um Hilfsfunktionen auszulagern und diese über Dependency-Injection dem `AuditingService`, `ExceptionLogger` und `Interceptor` zur Verfügung zu stellen.

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original