htlperg
innovativ | kreativ | praxisnah

Project Comparison

Automatisierter Vergleich von Softwareprojekten mit visueller Darstellung von Unterschieden

DIPLOMARBEIT

Höhere Abteilung für Informatik

01/07/2025 – 26/03/2026

Projektmitglieder: Lukas Langeder
Andreas Prinz

Betreuer: Dipl.-Ing. Helmut Otto

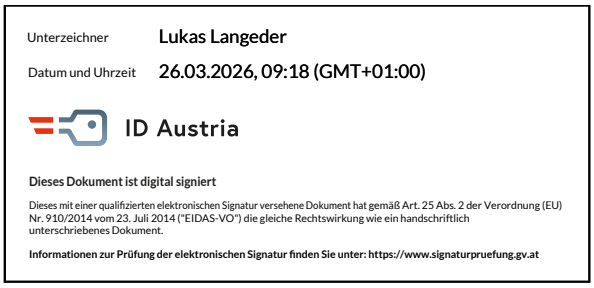
Eidesstattliche Erklärung

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

Bei der Erstellung der Arbeit haben wir die generativen KI-Tools ChatGPT und DeepL Write zu folgendem Zweck verwendet: Rechtschreib- und Grammatikprüfung, Formulierungshilfen zur sprachlichen Verbesserung und Umformulierung einzelner Textpassagen, Generierung von Code-Kommentaren.

Perg, 26.03.2026

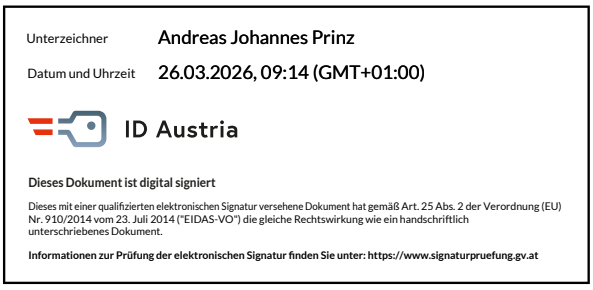
Ort, Datum



Unterschrift, Lukas Langeder

Perg, 26.03.2026

Ort, Datum



Unterschrift, Andreas Prinz

Gendererklärung

Im Sinne der besseren Lesbarkeit werden in dieser Diplomarbeit personenbezogene Bezeichnungen, die sich zugleich auf Frauen und Männer beziehen, generell nur in der im Deutschen üblichen maskulinen Form angeführt. Dies soll jedoch keinesfalls eine Geschlechterdiskriminierung oder eine Verletzung des Gleichheitsgrundsatzes zum Ausdruck bringen.

Danksagung

Wir möchten uns herzlich bei sämtlichen Lehrkräften der HTL Perg sowie bei der Firma ENGEL bedanken, die uns bei der Umsetzung unserer Diplomarbeit unterstützt haben.

Ein besonderer Dank gilt unserem Diplomarbeitsbetreuer Herrn Dipl.-Ing. Helmut Otto, der uns insbesondere beim Verfassen unserer schriftlichen Arbeit mit wertvollen Anregungen, Feedback und seiner ständigen Bereitschaft zur Unterstützung begleitet hat.

Ebenso möchten wir uns bei unserem Projektpartner ENGEL sowie besonders bei unseren Projektbetreuern Stefan Wallner und Hannes Windischhofer bedanken. Durch ihre fachliche Begleitung, ihre Unterstützung bei organisatorischen und technischen Fragestellungen sowie den regelmäßigen Austausch haben sie maßgeblich zum erfolgreichen Verlauf des Projekts beigetragen und die Umsetzung unserer Diplomarbeit wesentlich unterstützt.

Kurzfassung

Im Rahmen dieser Diplomarbeit wurde in Zusammenarbeit mit ENGEL eine Webanwendung entwickelt, die den Vergleich zweier Softwareprojekte auf strukturierte und übersichtliche Weise ermöglicht. Die Anwendung ist als Erweiterung des bestehenden Systems Control Studio gedacht.

Ziel des Projekts ist es, Unterschiede zwischen zwei Softwareprojekten automatisch zu erkennen und dem Benutzer übersichtlich darzustellen. Dabei werden Änderungen auf Datei- und Ordnerebene erfasst und visuell hervorgehoben. Dadurch wird die Analyse von Änderungen deutlich erleichtert, insbesondere bei großen und komplexen Projektstrukturen, bei denen ein manueller Vergleich sehr aufwendig wäre.

Die entwickelte Lösung wurde als Fullstack-Anwendung umgesetzt und besteht aus einem Angular-Frontend sowie einem C# .NET-Backend. Die Kommunikation erfolgt über eine REST-Schnittstelle. Über die Benutzeroberfläche können zwei Projekte ausgewählt und die Vergleichsergebnisse in verschiedenen Ansichten dargestellt werden. Dazu gehören unter anderem Baumstrukturen, Tabellen und Detailansichten, die eine genaue Analyse der Unterschiede ermöglichen.

Das Backend übernimmt die Verarbeitung der Projektdaten und führt den eigentlichen Vergleich durch. Dabei werden nicht nur strukturelle Unterschiede, sondern auch Metadaten sowie SAP-bezogene Informationen berücksichtigt. Die aufbereiteten Ergebnisse werden anschließend an das Frontend übergeben und dort verständlich visualisiert.

Zusammenfassend erleichtert die entwickelte Anwendung den Vergleich von Softwareprojekten erheblich, reduziert den manuellen Aufwand und unterstützt die Benutzer dabei, Änderungen schneller und zuverlässiger zu erkennen.

Abstract

This diploma thesis presents the development of a web application for comparing two software projects in a clear and structured way. The project was carried out in cooperation with ENGEL and is integrated into the existing software environment Control Studio.

The purpose of the application is to help users identify differences between two projects more quickly and easily. Instead of comparing files manually, the system automatically detects changes in folders and files and displays them in a clear visual form. This is especially helpful for larger projects, where manual comparison is time-consuming and difficult to manage.

The product was developed as a full-stack application with an Angular frontend and a C# .NET backend. The frontend allows users to select projects and view the comparison results in different forms, such as tree structures, tables, and detailed views. The backend performs the comparison and prepares the data for display.

In addition to file and folder differences, the application also considers metadata and SAP-related information. This gives users a more complete overview of the changes between projects.

In summary, the developed solution improves the comparison of software projects, reduces manual work, and helps users analyze project changes more efficiently.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Motivation . . . 1
1.2 Ausgangslage . . . 1
1.3 Zielsetzung . . . 2
1.4 Projektinhalt . . . 2
1.5 Projektumfeld . . . 3

2 Grundlagen und Methoden **5**

2.1 Grundlegende Fachbegriffe . . . 5
2.2 Verwendete Technologien . . . 11
2.3 Verwendete Entwicklungssysteme . . . 16
2.4 Bibliotheken und Plug-Ins . . . 21

3 Planung und Realisierung **28**

3.1 Projektorganisation . . . 28
3.2 Meilensteine . . . 28

4 Gesamtarchitektur **30**

4.1 Funktionale und Nicht-funktionale Anforderungen . . . 30
4.2 Technischer Überblick . . . 32
4.3 Softwarearchitektur . . . 34
4.4 Projektstruktur . . . 36

5 Implementierung **43**

5.1 Webanwendung . . . 43
5.2 Backend . . . 59

6 Ergebnis **84**

6.1 Projektauswahl . . . 84
6.2 Übersichtsansicht . . . 84
6.3 Baumstrukturansicht (TreeView) . . . 85

6.4 Dateivergleich (FileView) . . . 86
6.5 SAP-Ansicht . . . 87
6.6 Backend . . . 88

7 Resümee **89**

8 Aufgabenverteilung **91**
8.1 Lukas Langeder . . . 91
8.2 Andreas Prinz . . . 93

Glossar **V**

Literaturverzeichnis **VII**

Abbildungsverzeichnis **XIII**

Anhang **XIV**
A API-Dokumentation . . . XIV
B Plakat . . . XXIV
C Praxisteildokumentation – Lukas Langeder . . . XXVI
D Praxisteildokumentation – Andreas Prinz . . . XXXII

1 Einleitung

1.1 Motivation

Die zunehmende Komplexität von Softwareprojekten stellt Unternehmen vor die Herausforderung, Änderungen zwischen verschiedenen Projektständen effizient zu erkennen und nachvollziehbar darzustellen. Besonders bei umfangreichen Projektstrukturen ist ein manueller Vergleich mit großem Zeitaufwand verbunden und kann schnell unübersichtlich werden.

Auch bei der Firma ENGEL bestand der Bedarf, Softwareprojekte gezielt und strukturiert miteinander zu vergleichen. Dadurch entstand die Idee, eine Anwendung zu entwickeln, die diesen Prozess automatisiert und die Unterschiede übersichtlich visualisiert.

Unsere Ausbildung an der HTL Perg vermittelt uns die notwendigen Kenntnisse in den Bereichen Softwareentwicklung, Webtechnologien und systematischer Problemlösung. Besonders motivierend war für uns die Möglichkeit, gemeinsam mit einem Unternehmen an einer praxisnahen Aufgabenstellung zu arbeiten und eine Lösung mit direktem Anwendungsbezug zu entwickeln.

1.2 Ausgangslage

Im bestehenden System „Control Studio“, einem globalen Workbench-Tool von ENGEL, gab es bisher keine zentrale Möglichkeit, zwei Softwareprojekte strukturiert miteinander zu vergleichen und Unterschiede übersichtlich darzustellen.

Der Vergleich von Projekten war daher mit manuellem Aufwand verbunden und bei größeren Projektstrukturen nur schwer nachvollziehbar. Insbesondere Änderungen in Dateien und Ordnern konnten nicht schnell und einheitlich analysiert werden.

Ausgehend von dieser Problemstellung wurde im Rahmen der Diplomarbeit eine Anwendung entwickelt, welche den Vergleich zweier Projekte automatisiert und die Unterschiede strukturiert sowie visuell aufbereitet darstellt.

1.3 Zielsetzung

Ziel des Projekts ist es, eine Anwendung zu entwickeln, die den Vergleich zweier Softwareprojekte auf strukturierte und übersichtliche Weise ermöglicht. Dabei sollen Unterschiede auf Datei- und Ordnerebene automatisch erkannt und dem Benutzer klar dargestellt werden, sodass eine effiziente Analyse von Änderungen möglich ist.

Mit der entwickelten Anwendung soll es möglich sein, zwei Projekte direkt miteinander zu vergleichen und deren Struktur sowie Inhalte visuell aufzubereiten. Unterschiede sollen dabei farblich hervorgehoben werden, um eine schnelle Orientierung zu ermöglichen. Zusätzlich soll der Benutzer detaillierte Einblicke in einzelne Dateien erhalten und Unterschiede zeilenweise analysieren können.

Das finale Produkt der Diplomarbeit ist eine Fullstack-Anwendung, bestehend aus einem Angular-Frontend und einem C#-Backend. Die Kommunikation erfolgt über eine REST-Schnittstelle.

1.4 Projektinhalt

1.4.1 Frontend

Die Schnittstelle zum Benutzer bildet das Frontend der Anwendung. Über diese Benutzeroberfläche ist es möglich, zwei Softwareprojekte auszuwählen und den Vergleich zu starten.

Die Anwendung stellt die Ergebnisse in verschiedenen Ansichten dar, darunter eine Baumstruktur zur Darstellung der Ordner und Dateien sowie Tabellen- und Detailansichten für weiterführende Informationen. Unterschiede zwischen den Projekten werden visuell hervorgehoben, um eine schnelle Orientierung zu ermöglichen.

Zusätzlich bietet das Frontend verschiedene Interaktionsmöglichkeiten, wie das Filtern der angezeigten Daten oder das Öffnen einzelner Dateien zur detaillierten Analyse.

1.4.2 Backend

Das Backend übernimmt den eigentlichen Vergleich der ausgewählten Softwareprojekte. Es verarbeitet die übergebenen Daten und vergleicht die Projekte sowohl auf Metadaten- als auch auf SAP-Ebene. Dabei werden Unterschiede auf Ordner- und Dateiebene erkannt.

Es werden sowohl hinzugefügte, fehlende als auch geänderte Dateien identifiziert. Zusätzlich können Inhalte einzelner Dateien verglichen werden, um Änderungen im Detail darzustellen.

Die aufbereiteten Ergebnisse werden dem Frontend zur Verfügung gestellt, damit sie dort übersichtlich angezeigt und weiterverarbeitet werden können.

1.5 Projektumfeld

1.5.1 Projektteam

Das Projektteam besteht aus Andreas Prinz und Lukas Langeder. Die genaue Aufteilung der Aufgaben findet sich weiter unten.

Abbildung 1: Projektteam: v. l. n. r. Andreas Prinz und Lukas Langeder

1.5.2 Auftraggeber

Auftraggeber dieser Diplomarbeit ist die Firma ENGEL Austria GmbH (Logo Abbildung 2) mit Sitz in Schwertberg. Das Unternehmen ist international im Bereich Maschinenbau tätig und entwickelt und produziert unter anderem Spritzgussmaschinen sowie Softwarelösungen zur Unterstützung von Produktions- und Engineeringprozessen.

Abbildung 2: ENGEL Logo

Die Zusammenarbeit mit dem Auftraggeber erfolgte während der gesamten Projektlaufzeit in enger Abstimmung. Dabei konnten fachliche Anforderungen und technische Fragestellungen direkt mit dem Unternehmen geklärt werden. Die Anwendung ist als Erweiterung von „Control Studio“ vorgesehen, wurde im Rahmen der Diplomarbeit jedoch zunächst als eigenständige Lösung entwickelt. Die spätere Integration in das bestehende System erfolgt durch ENGEL. [1]

1.5.3 Betreuung

Die Diplomarbeit wurde im schulischen Rahmen von Herrn Dipl.-Ing. Helmut Otto betreut. Während der Planungsphase sowie insbesondere bei der Dokumentation des Projekts stand das Projektteam in regelmäßigem Austausch mit ihm.

Durch seine Unterstützung sowie das kontinuierliche Feedback konnten technische und organisatorische Herausforderungen effizient bewältigt werden. Die Betreuung trug wesentlich zur erfolgreichen Umsetzung der Diplomarbeit bei.

2 Grundlagen und Methoden

Um die praktische Umsetzung nachvollziehen zu können, sind theoretische Grundlagen sowie methodische Konzepte erforderlich. Daher werden grundlegende Fachbegriffe, verwendete Technologien, Entwicklungssysteme sowie Bibliotheken und Plug-ins erläutert.

2.1 Grundlegende Fachbegriffe

Relevante Fachbegriffe der Webentwicklung und deren Bedeutung für die Anwendung werden dargestellt. Dabei werden wichtige Konzepte wie DOM, Frameworks, Komponenten, Services, DTOs, Unit-Tests, REST und Interfaces behandelt.

2.1.1 DOM

In den Anfangsjahren des World Wide Web bestanden Webseiten hauptsächlich aus statischen HTML-Dokumenten. Inhalte und Struktur waren fest definiert und konnten nach dem Laden im Browser kaum verändert werden. Mit zunehmendem Bedarf an Interaktivität und Benutzerfreundlichkeit entstand der Bedarf, Webseiten dynamisch anpassen zu können, ohne dass sie vollständig neu geladen werden.

Um diese Dynamik zu gewährleisten, wurde DOM eingeführt. Es stellt die Struktur eines HTML-Dokuments in Form einer Baumstruktur dar, in der jedes Element als Objekt repräsentiert wird. Dadurch ist es möglich, gezielt auf einzelne Bestandteile einer Webseite zuzugreifen und diese dynamisch zu verändern.

Mit der zunehmenden Komplexität moderner Webanwendungen wurde jedoch deutlich, dass der direkte Zugriff auf das DOM schnell unübersichtlich und fehleranfällig werden kann. Entwickler mussten viel zusätzlichen Code schreiben, um Änderungen im Programmzustand korrekt in der Benutzeroberfläche darzustellen. Aus diesem Grund wurden Frameworks entwickelt, die wiederkehrende Aufgaben vereinfachen und eine klare Struktur für die Entwicklung vorgeben. [2]

Aus dieser Entwicklung heraus ergibt sich als nächster zentraler Begriff das Framework, das in modernen Webanwendungen eine grundlegende Rolle spielt.

2.1.2 Framework

Ein Framework ist ein vorgefertigtes Software-Grundgerüst, das Entwicklern eine strukturierte Basis für die Entwicklung von Anwendungen bietet. Es stellt wichtige Funktionen und Bausteine bereit, die in vielen Projekten benötigt werden. Dadurch kann Entwicklungsaufwand reduziert und die Wartbarkeit der Software verbessert werden.

Frameworks geben meist den grundlegenden Ablauf der Anwendung vor und Entwickler fügen dann ihre eigene Logik in die vom Framework vorgegebenen Strukturen ein. Dieses Vorgehen sorgt für einheitlich aufgebaute Projekte und erleichtert die Zusammenarbeit im Team.

Moderne Web-Frameworks stellen zahlreiche Mechanismen bereit, die die Entwicklung strukturierter und wartbarer Anwendungen unterstützen. Dazu gehören unter anderem komponentenbasierte Architekturen, Datenbindung oder Routing. Diese Konzepte helfen dabei, auch umfangreiche Benutzeroberflächen übersichtlich zu organisieren. [3]

Ein wichtiges Konzept moderner Web-Frameworks ist dabei die Aufteilung der Benutzeroberfläche in sogenannte Komponenten.

2.1.3 Komponenten

Komponenten bilden die Grundlage für den Aufbau von Benutzeroberflächen in Angular-Anwendungen. Sie ermöglichen es, die Oberfläche in kleinere, logisch getrennte Teile zu unterteilen, die jeweils eine bestimmte Aufgabe erfüllen. Dadurch bleibt die Anwendung übersichtlich und leichter wartbar.

Eine Komponente besteht aus einer TypeScript-Klasse, die die Logik enthält, sowie einem HTML-Template, das die Darstellung definiert. Zusätzlich können eigene Styles verwendet werden, um das Erscheinungsbild anzupassen. Die Definition erfolgt in Angular über den Decorator `@Component`. [4, 5]

Komponenten können ineinander verschachtelt werden. So entsteht eine hierarchische Struktur, bei der größere Ansichten aus mehreren kleineren Bausteinen zusammengesetzt sind. Dies hilft dabei, Code wiederzuverwenden und die Anwendung übersichtlich zu halten. [6]

Um Komponenten möglichst übersichtlich zu halten, wird fachliche Logik in der Regel nicht direkt in ihnen umgesetzt. Stattdessen wird diese in Services ausgelagert.

2.1.4 Services

Services dienen dazu, fachliche Logik und gemeinsam benötigte Daten zentral in einer Anwendung bereitzustellen. Durch die Auslagerung solcher Funktionalitäten aus den Komponenten wird eine klare Trennung der Verantwortlichkeiten (Separation of Concerns [7]) erreicht, was die Wartbarkeit und Wiederverwendbarkeit des Codes verbessert.

In Angular sind Services in der Regel gewöhnliche TypeScript-Klassen, die mit dem Decorator `@Injectable` gekennzeichnet werden. Dadurch kann das Framework den Service automatisch bereitstellen und bei Bedarf in Komponenten oder andere Services einbinden.

Typische Einsatzbereiche sind zum Beispiel API-Aufrufe, gemeinsam genutzte Daten, Authentifizierung oder auch Fehlerbehandlung. Oft werden Services so eingerichtet, dass sie in der ganzen Anwendung verfügbar sind und meistens nur eine Instanz davon existiert. Das hilft, den Code übersichtlich zu halten und die Anwendung besser zu strukturieren. [8]

Neben der strukturierten Kapselung von Logik durch Services spielt auch die klare Organisation der übertragenen Daten eine wichtige Rolle. Insbesondere bei der Kommunikation zwischen Frontend und Backend ist es erforderlich, Daten in einer kontrollierten und stabilen Form zu übermitteln. In diesem Zusammenhang kommen sogenannte DTOs zum Einsatz.

2.1.5 DTOs

In den frühen Phasen der Softwareentwicklung wurden Daten zwischen verschiedenen Anwendungsschichten häufig direkt über interne Objekte ausgetauscht. Insbesondere in verteilten Systemen führte dieser Ansatz jedoch rasch zu Problemen: Änderungen an der internen Datenstruktur hatten direkte Auswirkungen auf andere Komponenten, was die Wartbarkeit und Erweiterbarkeit der Systeme erheblich erschwerte. [9]

Mit dem zunehmenden Einsatz von mehrschichtigen Architekturen sowie verteilter Anwendungen entstand daher der Bedarf nach einer klaren Trennung zwischen internen Datenmodellen und externen Schnittstellen. In diesem Kontext wurden sogenannte Data Transfer Objects eingeführt. Sie dienen als einfache, strukturierte Container für Daten, die zwischen unterschiedlichen Schichten oder Systemen übertragen werden. [10]

DTOs bestehen in der Regel ausschließlich aus Attributen und enthalten keine Geschäftslogik. Ihr primärer Zweck liegt in der effizienten und kontrollierten Übertragung von Daten. Durch den Einsatz von DTOs wird die Kopplung zwischen Frontend und Backend reduziert, da externe

Komponenten nicht direkt auf interne Modelle zugreifen müssen [11]. Dies ermöglicht es, interne Änderungen vorzunehmen, ohne bestehende Schnittstellen zu beeinträchtigen.

Ein weiterer Vorteil von DTOs liegt in der gezielten Auswahl der zu übertragenden Daten. Es können ausschließlich jene Attribute bereitgestellt werden, die für den jeweiligen Anwendungsfall notwendig sind. Dadurch ergeben sich sowohl Sicherheitsvorteile, etwa durch das Verbergen sensibler Informationen und den Schutz vor Over-Posting-Schwachstellen, als auch Performancevorteile durch reduzierte Datenmengen. [12]

In modernen Webanwendungen haben sich DTOs als etablierter Standard in der Kommunikation zwischen Backend und Frontend durchgesetzt. Insbesondere in Verbindung mit REST- oder API-basierten Architekturen bilden sie eine zentrale Grundlage für saubere, wartbare und skalierbare Softwarelösungen. Moderne Frameworks unterstützen heute häufig die automatische Erstellung, Serialisierung und Validierung der übertragenen Datenmodelle, wodurch ihre Integration weiter vereinfacht wurde. [13]

Während DTOs vor allem die Struktur und den sicheren Austausch von Daten zwischen Anwendungsschichten definieren, ist es ebenso entscheidend, die korrekte Funktionalität der zugrunde liegenden Programmlogik sicherzustellen. Hier setzen Unit-Tests an, die eine systematische und automatisierte Überprüfung einzelner Softwarebestandteile ermöglichen.

2.1.6 Unit-Tests

Vor der Etablierung automatisierter Testverfahren wurden Programme häufig manuell getestet, indem komplette Anwendungen ausgeführt und deren Verhalten überprüft wurde. Dieser Ansatz war jedoch zeitaufwendig, fehleranfällig und kaum skalierbar, insbesondere mit zunehmender Größe und Komplexität der Softwareprojekte. Fehler konnten oft erst spät im Entwicklungsprozess erkannt werden, was den Aufwand für Korrekturen erheblich erhöhte.

Um diesem Problem entgegenzuwirken, wurden Unit-Tests eingeführt. Dabei handelt es sich um automatisierte Tests, mit denen einzelne, klar abgegrenzte Funktionseinheiten eines Programms – sogenannte Units – überprüft werden. Eine Unit kann beispielsweise eine Methode oder eine Funktion sein. Ziel von Unit-Tests ist es, diese Programmausschnitte unabhängig von anderen Komponenten zu testen und deren korrektes Verhalten sicherzustellen.

Unit-Tests werden typischerweise während oder vor der eigentlichen Implementierung erstellt und regelmäßig ausgeführt. Insbesondere beim sogenannten testgetriebenen Entwicklungsansatz (Test Driven Development) werden Tests vor der Implementierung geschrieben. Dabei wird zunächst definiert, welches Ergebnis eine bestimmte Funktion liefern soll, bevor der eigentliche

Programmcode entwickelt wird. Dieser Ansatz unterstützt eine strukturierte Entwicklung und erleichtert das frühzeitige Erkennen von Fehlern.

In modernen Softwareprojekten sind Unit-Tests ein zentraler Bestandteil des Entwicklungsprozesses. Sie tragen wesentlich zur Qualitätssicherung bei, erleichtern Refactorings und helfen, eine saubere und wartbare Softwarearchitektur zu gewährleisten. Durch die Automatisierung der Tests können Änderungen am Code zuverlässig überprüft werden, ohne bestehende Funktionalitäten unbeabsichtigt zu beeinträchtigen. [14, 15]

Neben der internen Qualitätssicherung spielt jedoch auch die Kommunikation zwischen verschiedenen Systemkomponenten eine entscheidende Rolle. Insbesondere bei verteilten Webanwendungen ist eine klar definierte und standardisierte Schnittstelle zwischen Client und Server erforderlich, welche im Folgenden anhand der RESTful-Architektur erläutert wird.

2.1.7 RESTful

In den Anfangsjahren verteilter Systeme erfolgte die Kommunikation zwischen Client und Server häufig über proprietäre Schnittstellen oder komplexe Protokolle. Diese Lösungen waren oft stark gekoppelt, schwer wartbar und nur eingeschränkt skalierbar. Mit der zunehmenden Verbreitung des World Wide Web entstand daher der Bedarf nach einer einheitlichen, einfachen und standardisierten Architektur für Webschnittstellen.

Als Antwort darauf wurde das Architekturkonzept Representational State Transfer eingeführt. REST beschreibt eine Reihe von Prinzipien zur Gestaltung verteilter Systeme, bei denen Ressourcen eindeutig über URLs identifiziert werden [16]. Auf diese Ressourcen wird mithilfe standardisierter HTTP-Methoden wie GET, POST, PUT und DELETE zugegriffen, um Daten abzurufen, zu erstellen, zu verändern oder zu löschen. [17]

Ein zentrales Merkmal von REST ist die Zustandslosigkeit (Statelessness). Jede Anfrage an den Server enthält alle notwendigen Informationen zur Verarbeitung, sodass serverseitig keine Sitzungsinformationen gespeichert werden müssen. Dies vereinfacht die Serverarchitektur erheblich und ermöglicht eine hohe Skalierbarkeit sowie eine klare Trennung zwischen Client und Server. [18]

In modernen Webanwendungen hat sich die RESTful-Architektur als etablierter Standard für die Umsetzung von Webservices durchgesetzt. RESTful-APIs bilden heute die Grundlage für die Kommunikation zwischen Frontend und Backend, insbesondere in Kombination mit Frameworks wie Angular oder anderen Single-Page-Anwendungen [11]. Durch ihre Einfachheit, Flexibilität

und breite Unterstützung sind sie ein wesentlicher Bestandteil moderner Softwarearchitekturen.

Um solche Architekturen langfristig wartbar und erweiterbar zu gestalten, ist jedoch nicht nur die externe Schnittstelle entscheidend, sondern auch die interne Struktur des Programmcodes. Ein zentrales Konzept zur Entkopplung von Implementierungen innerhalb der Anwendung stellen dabei Interfaces dar.

2.1.8 Interfaces

Mit dem Aufkommen der objektorientierten Programmierung wuchs die Komplexität von Softwareprojekten zunehmend. Klassen übernahmen immer mehr Verantwortlichkeiten, wodurch Abhängigkeiten zwischen einzelnen Komponenten stark anstiegen. Änderungen an einer konkreten Implementierung hatten häufig direkte Auswirkungen auf andere Teile der Anwendung, was die Wartbarkeit und Erweiterbarkeit erheblich erschwerte. [19]

Um diesem Problem entgegenzuwirken, wurden Interfaces eingeführt. Interfaces definieren, welche Methoden und Attribute eine Klasse bereitstellen muss, ohne dabei die konkrete Implementierung vorzugeben. Sie beschreiben somit die Funktionalität eines Programmteils, nicht jedoch dessen interne Umsetzung. Klassen, die ein Interface implementieren, sind verpflichtet, alle darin definierten Elemente bereitzustellen. [19]

Der Einsatz von Interfaces ermöglicht eine klare Entkopplung zwischen der Definition einer Funktionalität und ihrer Umsetzung. Dadurch kann eine konkrete Implementierung jederzeit ausgetauscht oder erweitert werden, ohne dass abhängige Komponenten angepasst werden müssen. Dies erhöht die Flexibilität der Softwarearchitektur und unterstützt eine saubere Trennung der Verantwortlichkeiten (Separation of Concerns [7]). [20]

In modernen, serviceorientierten Architekturen spielen Interfaces eine zentrale Rolle. Sie erleichtern die Wartung und Weiterentwicklung von Software und unterstützen zudem das automatisierte Testen. Durch den Einsatz von Interfaces können reale Implementierungen im Testfall problemlos durch Mock-Objekte oder simulierte Klassen ersetzt werden, wodurch Unit-Tests unabhängig und reproduzierbar durchgeführt werden können. [15]

Nachdem damit zentrale Grundbegriffe erläutert wurden, werden im nächsten Abschnitt die konkret eingesetzten Technologien vorgestellt.

2.2 Verwendete Technologien

Die im Projekt verwendeten Technologien und deren wesentliche Eigenschaften werden erläutert. Dabei werden im Frontend Node.js, NPM und Angular sowie im Backend C# .NET, ASP.NET und SAP und deren Zusammenspiel betrachtet.

2.2.1 Frontend

NodeJS

Node.js (Logo Abbildung 3) ist eine serverseitige JavaScript-Laufzeitumgebung, die von Ryan Dahl entwickelt wurde. Sie ermöglicht es, JavaScript auch außerhalb des Browsers auszuführen, beispielsweise für Webserver und Netzwerk-Anwendungen. Ein wichtiges Merkmal von Node.js ist die asynchrone, ereignisgesteuerte Arbeitsweise. Das System kann dadurch viele Anfragen gleichzeitig bearbeiten, ohne auf einzelne Vorgänge warten zu müssen, was zu einer hohen Effizienz und Skalierbarkeit führt. [22]

Abbildung 3: Node.js Logo [21]

Zur Verwaltung von Erweiterungen und Bibliotheken wird der Paketmanager npm verwendet.

NPM

NPM (Logo Abbildung 4) ist der standardmäßige Paketmanager für Node.js und wurde ursprünglich von Isaac Z. Schlueter entwickelt. Es dient zur Verwaltung von Bibliotheken und Abhängigkeiten in JavaScript-Projekten, damit lassen sich externe Pakete einfach installieren, aktualisieren und entfernen.

Abbildung 4: NPM Logo [23]

Ein zentrales Element von NPM ist die Datei `package.json`. Dort werden wichtige Projektinformationen sowie benötigte Abhängigkeiten definiert. Dadurch lassen sich Projekte reproduzierbar einrichten und verwalten.

Bei der Installation von Paketen legt NPM automatisch den Ordner `node_modules` im Projektverzeichnis an. In diesem Ordner werden alle installierten Abhängigkeiten sowie deren Unterabhängigkeiten gespeichert.

NPM greift auf ein umfangreiches Online-Repository zu, das eine große Anzahl frei verfügbarer Open-Source-Pakete bereitstellt. Dadurch müssen häufig benötigte Funktionen nicht selbst implementiert werden. [24]

Typischerweise werden Angular-Projekte über NPM verwaltet und um zusätzliche Pakete erweitert. Im Folgenden wird daher Angular für die Frontend-Entwicklung beschrieben.

Angular

Angular (Logo Abbildung 5) ist ein clientseitiges webbasiertes Framework und besonders für SPAs geeignet. Es wird von Google entwickelt und gepflegt und dient der strukturierten Umsetzung komplexer Frontend-Anwendungen. Das Framework entstand aus dem früheren Projekt AngularJS, es wurde grundlegend neu konzipiert und wird heute weiterentwickelt.

Abbildung 5: Angular Logo [25]

Angular basiert hauptsächlich auf TypeScript und verfolgt eine komponentenbasierte Architektur. Die Benutzeroberfläche wird in wiederverwendbare Komponenten aufgeteilt, welche jeweils für einen bestimmten Teil der Anwendung zuständig sind. Zudem kommen Services zum Einsatz, um gemeinsame Logik zentral bereitzustellen und Komponenten zu entlasten. [26]

Angular bietet eine automatische Datenbindung. Das bedeutet, Änderungen zwischen Benutzeroberfläche und Programmcode werden automatisch synchronisiert. Außerdem verfügt Angular über ein integriertes Routing-System. Es ermöglicht die Navigation zwischen verschiedenen Ansichten innerhalb einer SPA. [27]

Die Angular CLI ermöglicht es, Projekte automatisiert zu erstellen, Komponenten und Services zu generieren. Sie wird unter anderem auch benutzt, um Anwendungen zu bauen und lokal auszuführen. [28]

Nach den für das Frontend verwendeten Technologien werden im nächsten Schritt die Technologien des Backends beschrieben.

2.2.2 Backend

C# .NET

C# .NET (Logo Abbildung 6) ist ein Framework, welches meist für moderne Webanwendungen und Desktopsoftware verwendet wird. Es gibt zwei Arten: .NET Framework und .NET Core. .NET Framework wurde vor allem früher verwendet. Heutzutage wurde es weitgehend durch .NET Core ersetzt und wird meist nur noch für ältere Projekte genutzt. [30]

Abbildung 6: .NET Logo [29]

Die .NET-Plattform wurde Anfang der 2000er Jahre von Microsoft entwickelt. Die erste Version von .NET Framework wurde im Jahr 2002 veröffentlicht. Ziel war es, eine moderne Entwicklungsplattform zu schaffen, mit der unterschiedliche Arten von Anwendungen mit einer gemeinsamen Laufzeitumgebung entwickelt werden können. [31]

Ursprünglich war .NET Framework ausschließlich für das Betriebssystem Windows ausgelegt. Mit dem steigenden Bedarf an plattformübergreifenden Programmen begann Microsoft später mit der Entwicklung von .NET Core. Dies wurde erstmals 2016 veröffentlicht und ermöglicht eine schnelle Ausführung von Anwendungen nicht nur unter Windows, sondern auch unter Linux und macOS. Seit der Veröffentlichung von .NET 5 im Jahr 2020 wurden .NET Framework und .NET Core schrittweise zu einer gemeinsamen Plattform zusammengeführt. Durch die Entwicklung von Microsoft ist auch das Deployment in der Microsoft Azure Cloud sehr einfach. [32]

Als Programmiersprache wird C# verwendet. Diese ist objektorientiert und typsicher. Dadurch ermöglicht sie eine gut strukturierte und wartbare Umsetzung von komplexen Anwendungen und unterstützt moderne Konzepte wie Asynchronität, Dependency Injection und eine starke Typisierung. [33, 34, 20]

Ein Hauptbestandteil von .NET ist die CLR. Sie ist die Laufzeitumgebung, die für die Ausführung des Codes verantwortlich ist. Der geschriebene Code wird zunächst in eine Intermediate Language umgewandelt, also in eine Sprache zwischen Programmcode und Maschinencode. [35] Anschließend wird dieser Code vom JIT-Compiler in Maschinencode übersetzt, der dann direkt ausgeführt werden kann. Die CLR übernimmt außerdem die Speicherverwaltung, also das Entfernen von nicht mehr benötigten Objekten aus dem Speicher. Zusätzlich sorgt sie für die Typprüfung sowie die Fehlerprüfung, wodurch aussagekräftige Fehlermeldungen erzeugt werden. [36]

Auf Basis der .NET-Plattform können unterschiedliche Arten von Anwendungen entwickelt werden, beispielsweise Desktopprogramme, Cloudanwendungen oder Webanwendungen.

Auf dieser Grundlage baut auch ASP.NET auf, das speziell für Webanwendungen und Webschnittstellen verwendet wird.

ASP.NET

Für die Entwicklung moderner Webanwendungen stellt .NET das Framework ASP.NET Core bereit, welches speziell für serverseitige Websysteme und Web-APIs entwickelt wurde. Es wird vor allem verwendet, um serverseitige Anwendungen zu erstellen, die über das Internet erreichbar sind, beispielsweise Web-APIs oder dynamische Webseiten.

Die moderne Variante des Frameworks ist ASP.NET Core. Dabei handelt es sich um eine vollständig überarbeitete und modular aufgebaute Version von ASP.NET, welches als Open-Source-Projekt entwickelt wird. ASP.NET Core wurde erstmals im Jahr 2016 veröffentlicht und ist plattformübergreifend nutzbar. Anwendungen können somit nicht nur unter Windows, sondern auch unter Linux oder macOS ausgeführt werden.

Ein wesentliches Merkmal von ASP.NET Core ist die hohe Leistungsfähigkeit und Skalierbarkeit. Das Framework wurde speziell für moderne Webanwendungen entwickelt und kann laut unabhängigen Benchmarks eine sehr hohe Anzahl von Anfragen pro Sekunde verarbeiten. Dadurch eignet es sich besonders für Webservices und REST-APIs, die eine große Menge an Daten verarbeiten müssen.

ASP.NET Core vereint mehrere zuvor getrennte Technologien wie ASP.NET MVC und ASP.NET Web API in einem gemeinsamen Programmiermodell. Dadurch können sowohl Benutzeroberflächen als auch Webschnittstellen mit derselben Technologie entwickelt werden.

Ein weiterer Vorteil ist die modulare Architektur. Komponenten des Frameworks werden als einzelne Pakete bereitgestellt und können je nach Bedarf eingebunden werden. [37, 38]

Neben den direkt für die Softwareentwicklung eingesetzten Frontend- und Backend-Technologien spielte in dieser Arbeit auch das Unternehmenssystem SAP eine wichtige Rolle.

2.2.3 SAP

SAP steht für „Systeme, Anwendungen und Produkte in der Datenverarbeitung“. Das Unternehmen zählt heute zu den bekanntesten Anbietern von Unternehmenssoftware weltweit und gilt als größter Anbieter von Enterprise-Software [40]. Aufgrund dieser starken Marktposition ist SAP zudem das umsatzstärkste nicht-amerikanische Softwareunternehmen [41].

Abbildung 7: SAP Logo [39]

SAP wird vor allem zur Abbildung und Steuerung betrieblicher Geschäftsprozesse eingesetzt. Dazu gehören unter anderem die Verwaltung von Rohstoffen, Produktionskapazitäten, Bestellungen, Aufträgen sowie die Lohn- und Gehaltsabrechnung. Als ERP-System unterstützt SAP Unternehmen dabei, verschiedene Geschäftsbereiche in einem zentralen System zusammenzuführen und Daten durchgängig zu verwalten. [42]

Gegründet wurde SAP im Jahr 1972 von fünf ehemaligen IBM-Entwicklern aus Mannheim. 1973 erschien mit SAP R/1 die erste kommerzielle Lösung des Unternehmens. Darauf folgte 1979 SAP R/2, das den Funktionsumfang unter anderem um Bereiche wie Materialwirtschaft und Produktionsplanung erweiterte. Im Jahr 1992 wurde schließlich SAP R/3 veröffentlicht, das durch seine Client-Server-Architektur einen wichtigen technologischen Fortschritt darstellte. [43]

Im Laufe der Zeit wurden die Systeme kontinuierlich weiterentwickelt. Die aktuellste Generation ist SAP S/4HANA. Dieses System basiert auf der In-Memory-Datenbank SAP HANA und ermöglicht dadurch eine besonders schnelle Verarbeitung großer Datenmengen in Echtzeit. [44]

Neben klassischen ERP-Funktionen bietet SAP heute auch Lösungen für viele weitere Unternehmensbereiche an, beispielsweise für Customer-Relationship-Management, Supply-Chain-Management, Personalmanagement und Cloud-Anwendungen. Dadurch hat sich SAP von einem klassischen ERP-Anbieter zu einem breit aufgestellten Softwareunternehmen für unterschiedlichste Geschäftsprozesse entwickelt. [45]

Neben den verwendeten Technologien selbst sind für die praktische Umsetzung eines Softwareprojekts auch die eingesetzten Entwicklungssysteme von Bedeutung.

2.3 Verwendete Entwicklungssysteme

Relevante Entwicklungssysteme und deren Einsatz im Projekt werden beschrieben. Dabei werden Visual Studio Code, Visual Studio, Figma, Chrome DevTools, Git und Postman in ihrem jeweiligen Anwendungsbereich eingeordnet.

2.3.1 Visual Studio Code

Visual Studio Code (Logo Abbildung 8) ist ein plattformübergreifender Quellcode-Editor, der von Microsoft entwickelt und gepflegt wird. Der Editor wurde erstmals 2015 veröffentlicht.

Abbildung 8: VS Code Logo [46]

Es unterstützt eine Vielzahl von Programmiersprachen und bietet Funktionen wie Syntaxhervorhebung, IntelliSense-Autovervollständigung, integriertes Debugging. Zudem ist Git direkt integriert, wodurch die Versionskontrolle direkt innerhalb der Entwicklungsumgebung durchgeführt werden kann. Durch ein umfangreiches Erweiterungssystem kann die IDE flexibel an unterschiedliche Technologien und Frameworks angepasst werden, beispielsweise für Angular oder Node.js. [47]

Während für die Entwicklung des Frontends Visual Studio Code verwendet wurde, kam für die Umsetzung des Backends die Entwicklungsumgebung Visual Studio zum Einsatz.

2.3.2 Visual Studio

Visual Studio (Logo Abbildung 9) ist eine von Microsoft entwickelte Entwicklungsumgebung. Sie wird für die Erstellung von Desktopanwendungen, Webanwendungen, Webservices sowie mobilen Anwendungen verwendet. Da Visual Studio viele Tools für Microsoft-Technologien bereitstellt, wird es besonders häufig mit .NET verwendet.

Abbildung 9: Visual Studio Logo [48]

Ein zentrales Feature von Visual Studio ist der integrierte Editor. Dieser unterstützt Funktionen wie Syntaxhervorhebung, automatische Codevervollständigung und Refactoring. Dadurch wird das Schreiben von Programmcode erleichtert und die Produktivität bei der Entwicklung erhöht. Zusätzlich bietet Visual Studio einen integrierten Debugger, mit dem Programme schrittweise ausgeführt und

Fehler gezielt analysiert werden können. Dies ist insbesondere bei der Entwicklung komplexerer Anwendungen von großem Vorteil.

Neben der Bearbeitung von Code stellt Visual Studio auch zahlreiche weitere Tools zur Verfügung. Dazu gehören unter anderem Designer für grafische Benutzeroberflächen, Tools zur Datenbankanbindung sowie Funktionen zur Verwaltung von Projekten und Solutions. Außerdem kann die Entwicklungsumgebung durch Erweiterungen ergänzt werden, wodurch zusätzliche Programmiersprachen, Tools oder Schnittstellen eingebunden werden können. [49, 50]

Visual Studio wurde erstmals im Jahr 1997 veröffentlicht. Ziel war es, mehrere zuvor getrennte Entwicklungstools von Microsoft in einer gemeinsamen Umgebung zusammenzuführen. Im Laufe der Jahre wurde die Entwicklungsumgebung kontinuierlich erweitert und an neue Technologien angepasst. Insbesondere mit der Einführung von .NET gewann Visual Studio stark an Bedeutung. [51]

Für Entwickler stehen verschiedene Editionen von Visual Studio zur Verfügung. Besonders verbreitet ist die Community Edition, die kostenlos genutzt werden kann und sich vor allem an Einzelentwickler, Studierende sowie kleinere Teams richtet. Darüber hinaus existieren kostenpflichtige Varianten wie Professional und Enterprise, die zusätzliche Funktionen für professionelle Softwareentwicklung und Teamarbeit enthalten. [52]

Neben der Entwicklungsumgebung Visual Studio wurde auch das Design-Tool Figma verwendet, um die Benutzeroberfläche der Anwendung zu entwerfen und zu planen.

2.3.3 Figma

Figma (Logo Abbildung 10) ist ein webbasiertes Grafik- und Prototyping-Werkzeug, welches für digitale Designs von Websites und App-Oberflächen verwendet wird. Die Plattform ermöglicht es Teams, gemeinsam an Entwürfen zu arbeiten, da mehrere Benutzer gleichzeitig in Echtzeit an derselben Datei arbeiten können.

Abbildung 10: Figma Logo [53]

Figma ist webbasiert, daher erfolgt die Nutzung unabhängig vom Betriebssystem und ohne lokale Installation. [54]

Ergänzend zur Gestaltung der Benutzeroberfläche mit Figma wurden die Chrome DevTools zur Analyse, Fehlersuche und Optimierung der Anwendung verwendet.

2.3.4 Chrome DevTools

Die Chrome DevTools (Logo Abbildung 11) sind integrierte Entwicklerwerkzeuge des Webbrowsers Google Chrome, die von Google bereitgestellt werden. Sie werden zur Analyse, Entwicklung und Fehlerbehebung von Webanwendungen direkt im Browser verwendet. [56]

Abbildung 11: Chrome DevTools Logo [55]

Ein zentrales Feature ist das Elements Panel, mit dem die DOM-Struktur einer Webseite untersucht werden kann. Dort lassen sich HTML-Elemente inspizieren und CSS-Regeln temporär verändern, um Layout- und Darstellungsprobleme zu identifizieren. Außerdem ermöglicht dies ein Ausprobieren von Ideen. [57]

Die Console ist für die Ausgabe von Fehlermeldungen sowie das interaktive Ausführen von JavaScript-Code zuständig. Im Sources Panel können Breakpoints gesetzt werden, wodurch man den Programmablauf schrittweise analysieren kann. [58]

Zur Untersuchung der Netzwerkkommunikation steht das Network Panel zur Verfügung. Alle HTTP-/HTTPS-Anfragen und Informationen wie Ladezeiten, Statuscodes und Dateigrößen werden übersichtlich dargestellt und protokolliert. Dadurch lassen sich Performance-Probleme und fehlerhafte Requests leichter erkennen. [59]

Zur Analyse und Optimierung der Anwendung wurden die Chrome DevTools eingesetzt. Die Verwaltung des Quellcodes sowie die Zusammenarbeit im Team erfolgte über Git.

2.3.5 Git

Abbildung 12: Git Logo [60]

Git (Logo Abbildung 12) ist das am weitesten verbreitete Versionierungssystem in der Softwareentwicklung [61]. Der Name ist nicht eindeutig festgelegt und wird laut Dokumentation teilweise scherzhaft unterschiedlich interpretiert. Wenn man in positiver Stimmung ist, steht Git für „Global Information Tracker“, was sinngemäß mit „globaler Informationsverfolger“ übersetzt werden kann [62]. Git unterstützt Entwickler dabei, gemeinsam an einer Codebasis zu arbeiten, die als Repository bezeichnet wird. Dabei besitzt in der Regel jeder Entwickler eine lokale Kopie des gesamten Repositories einschließlich der Versionshistorie [63].

Ursprünglich wurde Git im Jahr 2005 von Linus Torvalds für die Versionsverwaltung des Linux-Kernels entwickelt [64]. Kurz nach der Veröffentlichung wurde die weitere Wartung an Junio Hamano übergeben, der das Projekt weiterführte [65]. Bis heute wird Git kontinuierlich weiterentwickelt und regelmäßig aktualisiert.

Ein zentrales Konzept von Git sind sogenannte Branches. Diese repräsentieren unterschiedliche Entwicklungsstände eines Projekts und ermöglichen es, Änderungen unabhängig voneinander vorzunehmen [66]. Dadurch können Entwickler parallel an verschiedenen Funktionen oder Korrekturen arbeiten, ohne die Hauptentwicklung unmittelbar zu beeinflussen. Üblicherweise wird für eine neue Aufgabe oder ein neues Feature ein eigener Branch verwendet.

Nach Abschluss der Arbeiten kann ein solcher Branch mit einem anderen Entwicklungszweig, meist dem Hauptbranch, zusammengeführt werden. Dieser Vorgang wird als Merge bezeichnet [66]. Auf diese Weise erleichtert Git die strukturierte Zusammenarbeit in Softwareprojekten. [67]

Neben der Versionsverwaltung wurde auch ein Werkzeug benötigt, mit dem sich Schnittstellen gezielt testen lassen. Hierfür wurde Postman eingesetzt.

2.3.6 Postman

Postman (Logo Abbildung 13) ist ein Programm zum Testen von APIs. Es wird vor allem verwendet, um HTTP-Anfragen an Schnittstellen zu senden und die Antworten übersichtlich anzuzeigen. Dadurch können Entwickler einfach überprüfen, ob einzelne Endpunkte richtig funktionieren. [69]

Abbildung 13: Postman Logo

[68]

Die Anwendung entstand im Jahr 2012 in Bangalore als Nebenprojekt von Abhinav Asthana. Er wollte damit das Testen von APIs vereinfachen, da bestehende Lösungen dafür oft umständlich waren. Anfangs wurde Postman kostenlos im Chrome Web Store veröffentlicht [70]. Später kamen Ankit Sobti und Abhijit Kane als Mitgründer dazu [71]. Im Jahr 2017 wurde der Hauptsitz des Unternehmens von Bangalore nach San Francisco verlegt. [72]

Mit Postman können Anfragen mit verschiedenen HTTP-Methoden wie `GET`, `POST`, `PUT` oder `DELETE` erstellt und ausgeführt werden. Zusätzlich lassen sich Header, Parameter, Authentifizierung und Request-Bodies festlegen. Die Antworten des Servers werden danach direkt angezeigt, wodurch Statuscodes, Rückgabewerte und Fehlermeldungen leichter kontrolliert werden können. [73]

Außerdem bietet Postman Funktionen für die Zusammenarbeit im Team. Dazu gehören zum Beispiel Workspaces, in denen mehrere Benutzer gemeinsam an APIs arbeiten können, sowie Collections, in denen zusammengehörige Anfragen gespeichert und strukturiert werden. [73]

Postman ist damit ein nützliches Tool bei der Entwicklung und beim Testen von Webschnittstellen. Vor allem bei REST-APIs erleichtert es die Überprüfung einzelner Endpunkte und hilft Fehler schneller zu finden. [69]

Aufbauend auf den Entwicklungssystemen werden im nächsten Abschnitt die zusätzlich verwendeten Bibliotheken und Plug-Ins beschrieben.

2.4 Bibliotheken und Plug-Ins

Die im Projekt verwendeten Bibliotheken und Plug-ins sowie deren grundlegende Eigenschaften werden beschrieben. Dabei werden im Frontend RxJS, Electron und Tailwind CSS sowie im Backend Swagger/OpenAPI, xUnit, NLog, ENGEL Libraries und der Myers-Differenzalgorithmus betrachtet.

2.4.1 Frontend

Im Frontend kamen neben den grundlegenden Technologien auch zusätzliche Bibliotheken zum Einsatz, die bestimmte Aufgaben gezielt unterstützen.

RxJS

RxJS (Logo Abbildung 14) ist eine Bibliothek zur Verarbeitung von asynchronen Datenströmen in JavaScript. Sie gehört zu den sogenannten Reactive Extensions, deren ursprüngliches Konzept von Microsoft entwickelt wurde. Ziel dieser Technologie war es, Ereignisse und Datenströme einheitlich zu behandeln und dadurch die asynchrone Programmierung zu vereinfachen. Auf Basis dieser Idee entstand später RxJS als Implementierung für JavaScript und entwickelte sich zu einem wichtigen Bestandteil moderner Webanwendungen.

Abbildung 14: RxJS Logo [74]

Die Bibliothek basiert auf dem Konzept der reaktiven Programmierung. Dabei werden Daten und Ereignisse als sogenannte Datenströme betrachtet, die über einen bestimmten Zeitraum hinweg Werte liefern können. Anwendungen können auf diese Werte reagieren, sobald sie verfügbar sind, was besonders bei Benutzerinteraktionen, Netzwerkkommunikation oder zeitabhängigen Prozessen von Vorteil ist.

Ein Hauptbestandteil von RxJS sind sogenannte Observables. Ein Observable stellt einen Datenstrom dar, der von anderen Programmteilen beobachtet werden kann. Komponenten oder Services können sich über eine sogenannte Subscription auf diesen Datenstrom registrieren und erhalten anschließend neue Werte, sobald diese verfügbar sind.

Ein Observable kann dabei drei verschiedene Arten von Ereignissen ausgeben:

- **next** – ein neuer Wert wird übertragen
- **error** – es tritt ein Fehler im Datenstrom auf

- **complete** – der Datenstrom wurde abgeschlossen

Neben Observables existieren in RxJS auch sogenannte Subjects. Ein Subject ist eine spezielle Form eines Observables, das zusätzlich selbst Werte erzeugen und an mehrere Empfänger gleichzeitig weitergeben kann. Es kann sowohl als Datenquelle als auch als Beobachter fungieren.

In Angular-Anwendungen werden Observables und Subjects häufig eingesetzt, um den Datenaustausch zwischen verschiedenen Teilen der Anwendung zu realisieren.

Darüber hinaus stellt RxJS eine Vielzahl von Operatoren zur Verfügung, mit denen Datenströme weiterverarbeitet werden können. Damit lassen sich eingehende Daten leichter verändern, filtern oder mit anderen Datenströmen kombinieren. Typische Beispiele sind die Operatoren `map`, mit dem Werte transformiert werden können, oder `filter`, mit dem nur bestimmte Daten aus einem Datenstrom weitergegeben werden.

Beispielsweise verwendet der HTTP-Client in Angular Observables, um Daten vom Backend abzurufen. Die Serverantworten werden dabei als Datenstrom bereitgestellt und können anschließend innerhalb der Anwendung weiterverarbeitet werden. [75, 76]

Die entwickelte Benutzeroberfläche basiert auf einer Angular-Webanwendung. Damit diese Anwendung jedoch als eigenständiges Desktopprogramm genutzt werden kann, wurde zusätzlich die Technologie Electron eingesetzt.

Electron

Electron (Logo Abbildung 15) wurde ursprünglich von der Firma GitHub entwickelt und im Jahr 2013 erstmals veröffentlicht. Das Framework entstand aus dem Projekt Atom Shell, das für den Code-Editor Atom entwickelt wurde. Das Ziel war es, eine Plattform zu schaffen, mit der sich Desktopprogramme einfach mit bekannten Webtechnologien erstellen lassen. Später wurde das Projekt in Electron umbenannt und als Open-Source-Framework veröffentlicht. [78]

Abbildung 15: Electron Logo [77]

Electron ist ein Framework zur Entwicklung von Desktop-Anwendungen mit Webtechnologien. Programme, welche mit JavaScript, HTML und CSS erstellt werden, können anschließend auf verschiedenen Betriebssystemen wie Windows, macOS und Linux ausgeführt werden.

Technisch basiert Electron auf einer Kombination aus Chromium und Node.js. Chromium wird für die Darstellung der grafischen Benutzeroberfläche verwendet, ähnlich wie bei einem Webbrowser. Node.js ermöglicht den Zugriff auf Funktionen des Betriebssystems, beispielsweise auf Dateien oder Prozesse. Dadurch können Anwendungen mit einer webbasierten Oberfläche erstellt werden, welche trotzdem wie klassische Desktopprogramme mit dem System interagieren. [79]

Für die Entwicklung der Benutzeroberfläche innerhalb dieser Electron-Anwendung werden moderne Webtechnologien eingesetzt. Die Gestaltung der Oberfläche mit CSS spielt eine wichtige Rolle. Um die grafische Oberfläche effizient und strukturiert umzusetzen, wird in dieser Arbeit das CSS-Framework TailwindCSS verwendet.

Tailwind CSS

TailwindCSS (Logo Abbildung 16) wurde im Jahr 2017 vom Entwickler Adam Wathan entwickelt und als Open-Source-Framework veröffentlicht. Die Idee hinter Tailwind entstand aus dem Wunsch, die Gestaltung von Benutzeroberflächen flexibler zu machen. Anstatt viele vorgefertigte Komponenten bereitzustellen, sollte ein Framework entstehen, das Entwicklern kleinere, kombinierbare CSS-Klassen zur Verfügung stellt, um Benutzeroberflächen einfacher und individueller zu gestalten. [81]

Abbildung 16: TailwindCSS Logo

[80]

TailwindCSS ist ein CSS-Framework zur Gestaltung moderner Benutzeroberflächen. Es basiert auf einem sogenannten Utility-First-Ansatz. Das bedeutet, dass viele kleine CSS-Klassen bereitgestellt werden, die jeweils eine bestimmte Eigenschaft definieren, beispielsweise Farben, Abstände, Schriftgrößen oder Layout-Eigenschaften. Diese Klassen können direkt im HTML-Code kombiniert werden, um das Design eines Elements zu bestimmen.

Ein weiterer Vorteil von TailwindCSS ist die hohe Anpassbarkeit. Über eine Konfigurationsdatei können beispielsweise Farben, Schriftarten oder Abstände zentral definiert werden. Dadurch zieht sich ein einheitliches Design quer durch die Anwendung. Änderungen am Design können an einer zentralen Stelle vorgenommen werden und wirken sich automatisch auf die gesamte Anwendung aus.

Zusätzlich verwendet TailwindCSS einen Build-Prozess, bei dem nur jene CSS-Klassen in die finale CSS-Datei übernommen werden, die im Projekt tatsächlich verwendet werden. Dadurch wird die Größe dieser Datei reduziert und die Ladezeit der Anwendung verbessert. [82]

Neben den Frontend-Bibliotheken kamen auch im Backend zusätzliche Werkzeuge und Bibliotheken zum Einsatz, die Entwicklung, Dokumentation und Wartung unterstützen.

2.4.2 Backend

Swagger / OpenAPI

Swagger (Logo Abbildung 17) ist eine Sammlung von Open-Source-Tools zur Entwicklung, Dokumentation und Nutzung von HTTP-Webservices bzw. REST-APIs. Dabei basiert Swagger auf dem Standard OpenAPI, mit dem Schnittstellen in einer strukturierten Form beschrieben werden können.

Abbildung 17: Swagger Logo [83]

In der Praxis wird Swagger häufig verwendet, um automatisch eine verständliche und interaktive Dokumentation für eine API zu erzeugen. Besonders verbreitet ist dabei die Swagger UI, eine webbasierte Oberfläche, über die Entwickler API-Endpunkte direkt testen und ausprobieren können. Dadurch kann der Entwicklungsprozess von APIs deutlich vereinfacht werden. [84]

Das Swagger-Projekt wurde im Jahr 2011 von Tony Tam gegründet, der als technischer Mitbegründer der Plattform Wordnik tätig war. Während der Entwicklung von Wordnik entstand der Bedarf nach einer automatisierten Dokumentation von APIs und der Generierung von Client-SDKs für den Zugriff auf die APIs. Um dieses Problem zu lösen, entwickelte Tam eine JSON-basierte Beschreibung von APIs, die auf die REST-Architektur aufbaut.

Im Laufe der Zeit entwickelte sich Swagger zu einem wichtigen Tool im Bereich der API-Entwicklung. Im Jahr 2015 wurde im Rahmen der Linux Foundation die OpenAPI-Initiative gegründet, welche die Weiterentwicklung der Spezifikation koordinieren sollte. Seit 2016 wird die ursprüngliche Swagger-Spezifikation offiziell als OpenAPI-Spezifikation bezeichnet, während der Name Swagger weiterhin für die zugehörigen Tools verwendet wird. [85]

Swagger unterstützt viele verbreitete Programmiersprachen, darunter Java, JavaScript, Scala oder C#. Dadurch lässt sich das Tool problemlos in unterschiedliche Entwicklungsumgebungen integrieren und wird häufig in modernen Webanwendungen eingesetzt.

Neben der Dokumentation und dem Testen von API-Endpunkten während der Entwicklung spielt auch die automatisierte Überprüfung von Software eine wichtige Rolle. Um die Funktiona-

lität einzelner Komponenten zuverlässig sicherzustellen, werden häufig Unit-Tests eingesetzt. [86]

xUnit

Ein verbreitetes Framework zum Schreiben und Ausführen von Unit-Tests für Programme, die mit .NET geschrieben wurden, ist xUnit (Logo Abbildung 18). Es wird verwendet, um einzelne Komponenten einer Applikation isoliert zu überprüfen. Dadurch können Fehler frühzeitig erkannt werden und die Qualität der Software verbessert werden.

Abbildung 18: xUnit Logo [87]

xUnit ist vollständig in der Programmiersprache C# implementiert und kann mit verschiedenen .NET-Laufzeitumgebungen, wie .NET Core und .NET Framework verwendet werden. Dadurch ist es möglich, Tests auf unterschiedlichen Betriebssystemen wie Windows, Linux oder macOS auszuführen.

Ein Test wird in einer eigenen Methode geschrieben und mit der Annotation `[Fact]` gekennzeichnet. Innerhalb dieser Methode können verschiedene Prüfungen durchgeführt werden. Die Rückgabewerte werden mithilfe von Assertions überprüft, um festzustellen, ob ein Programmabschnitt den erwarteten Wert liefert.

Neben automatisierten Tests spielt auch die Überwachung und Nachvollziehbarkeit von Anwendungen während der Ausführung eine wichtige Rolle. Insbesondere bei serverseitigen Anwendungen ist es wichtig, Fehler, Warnungen oder wichtige Systemereignisse zu protokollieren. Hierfür werden sogenannte Logging-Systeme eingesetzt. [88]

NLog

Ein häufig verwendetes Logging-Framework für .NET-Anwendungen ist NLog (Logo Abbildung 19). NLog ist eine Open-Source-Bibliothek, die es ermöglicht, Informationen über das Verhalten einer Anwendung während der Laufzeit zu protokollieren. Dabei können unterschiedliche Arten von Meldungen erfasst werden, beispielsweise Debug-, Informations-, Warn- oder Fehlermeldungen.

Abbildung 19: NLog Logo [89]

Die Bibliothek wurde ursprünglich von Jarek Kowalski entwickelt. Sie zeichnet sich durch eine hohe Flexibilität und eine einfache Konfiguration aus. Die Einstellungen für das Logging können

über Konfigurationsdateien vorgenommen werden, ohne dass Änderungen im Programmcode notwendig sind.

Ein zentrales Konzept von NLog ist die sogenannte Zielstruktur (Targets). Dabei kann festgelegt werden, wohin Logeinträge geschrieben werden sollen. Typische Ziele sind beispielsweise Logdateien, Konsolenausgaben oder Datenbanken. Zusätzlich können Regeln definiert werden, um zu bestimmen, welche Meldungen protokolliert werden und welche nicht.

Durch diese flexible Struktur eignet sich NLog besonders gut für größere Anwendungen, bei denen eine klare Nachvollziehbarkeit von Programmabläufen sowie eine strukturierte Fehleranalyse erforderlich sind. In vielen Projekten wird NLog daher eingesetzt, um Laufzeitinformationen zu erfassen und bei der Diagnose von Problemen zu unterstützen. [90, 91]

Ergänzend zu allgemein verfügbaren Bibliotheken wurden in dieser Arbeit auch firmeninterne Pakete verwendet, um bestehende Standards einzuhalten.

ENGEL Libraries

Um firmenweit einheitlichen Code zu gewährleisten, entwickelt und verwendet ENGEL eigene `Engel.Standard`-Pakete. Diese stellen unter anderem Funktionalitäten wie Logging, File-Operationen (z.B. das Auslesen und Schreiben von Dateien) sowie Methoden zur Verarbeitung von XML-Dateien bereit. Durch die Verwendung dieser Pakete wird sowohl die Konsistenz des Codes als auch die Geschwindigkeit der Programmierung erhöht. Zusätzlich erleichtert es auch die Wartung von bereits implementierten Funktionen.

Neben allgemeinen Entwicklungs- und Hilfsbibliotheken wurde für die konkrete fachliche Umsetzung auch ein spezieller Algorithmus benötigt, um Textunterschiede effizient zu erkennen.

Myers-Differenzalgorithmus

Um Unterschiede zwischen zwei Texten zu finden, wird häufig der Differenzalgorithmus von Eugene W. Myers verwendet. Dieser wurde 1986 veröffentlicht und ist der Hauptbestandteil hinter vielen Vergleichstools. Zum Beispiel basiert der Standard-Vergleichsalgorithmus von Git auf dem Myers-Verfahren. Ziel des Algorithmus ist es, eine Textsequenz mit einer minimalen Anzahl von Änderungen in eine andere zu überführen.

Als Beispiel werden die beiden Sequenzen $A = \text{ABCA}$ und $B = \text{ACA}$ miteinander verglichen. Zur Veranschaulichung kann der Vergleich zweier Sequenzen als sogenannter Edit-Graph dargestellt werden (siehe Abbildung 20). Dabei entsprechen diagonale Schritte übereinstimmenden Zeichen,

horizontale Schritte einer Löschoperation und vertikale Schritte einer Einfügeoperation. Der eingezeichnete Pfad zeigt eine mögliche Transformation der ersten Sequenz in die zweite. Der Algorithmus versucht somit, einen optimalen Pfad durch dieses Gitter zu finden und möglichst wenig Änderungen vorzunehmen. Gleichzeitig lässt sich daraus auch die längste gemeinsame Teilsequenz (Longest Common Subsequence) bestimmen, also der größte Teil des Textes, der in beiden Versionen identisch ist. Diese kann entlang der Diagonale im Graphen erkannt werden.

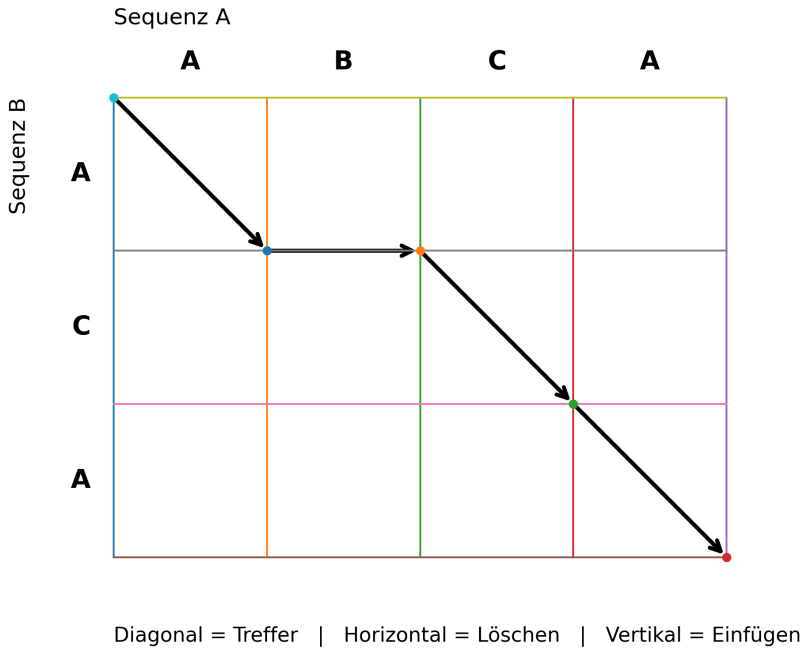


Abbildung 20: Vereinfachter Edit-Graph beim Vergleich zweier Sequenzen

Die Laufzeit beträgt $O(ND)$. Dabei beschreibt N die Gesamtlänge der beiden Sequenzen und D die minimale Anzahl an Einfüge- und Löschoperationen, die notwendig sind, um eine Textsequenz in die andere zu überführen. In der Praxis ist D häufig relativ klein, da sich verschiedene Versionen eines Textes meist nur geringfügig unterscheiden. Dadurch eignet sich der Algorithmus besonders gut für den Vergleich von Textdateien oder unterschiedlichen Versionen eines Dokuments. [92]

3 Planung und Realisierung

3.1 Projektorganisation

Die Planung und Umsetzung der Diplomarbeit erfolgte in Zusammenarbeit mit der Firma ENGEL. Zu Beginn der Arbeit wurden gemeinsam mit dem Diplomarbeitsbetreuer die grundlegenden Anforderungen, Aufgabenbereiche sowie zentrale Meilensteine festgelegt.

Während des Praktikums fand eine enge Zusammenarbeit mit dem Betreuer bei der Firma ENGEL statt. Innerhalb des Projektteams wurden die Aufgabenbereiche klar aufgeteilt:

- **Andreas Prinz**: Implementierung des Backends mit C# und .NET
- **Lukas Langeder**: Entwicklung des Angular-Frontends

Die Aufgaben für die jeweiligen Arbeitstage wurden regelmäßig im Team besprochen und abgestimmt. Durch diese Abstimmungen konnten auftretende Probleme frühzeitig erkannt und Anforderungen bei Bedarf präzisiert werden. Dadurch konnte der Projektfortschritt laufend überwacht und die Einhaltung der geplanten Meilensteine sichergestellt werden.

3.2 Meilensteine

Für die strukturierte Planung des Projekts wurden mehrere Meilensteine definiert. Diese markieren wichtige Etappen während der Umsetzung der Diplomarbeit und dienten gleichzeitig als Orientierung zur Kontrolle des Projektfortschritts.

- **30.06.2025**: Projektstart mit einer Einführung in das Thema sowie der Aufteilung der Aufgabenbereiche innerhalb des Teams (Frontend und Backend).
- **04.07.2025**: Abschluss der Anforderungsanalyse und Erstellung eines technischen Konzepts für die Umsetzung der Anwendung.
- **11.07.2025**: Beginn der Implementierung im Frontend und Backend sowie Definition der Schnittstellen zwischen den beiden Komponenten.
- **18.07.2025**: Entwicklung eines ersten funktionalen Prototyps, der den Vergleich von Projekten sowie die Darstellung der Ergebnisse im Frontend ermöglicht.

- **25.07.2025**: Abschluss der Implementierung, Durchführung von Tests sowie Abnahme der Anwendung durch den Betreuer.

4 Gesamtarchitektur

Die Gesamtarchitektur beschreibt die Struktur der Anwendung sowie das Zusammenspiel von Frontend, Backend und angebundenen Systemen. Dabei wird dargestellt, wie die einzelnen Komponenten miteinander kommunizieren und zusammenarbeiten.

4.1 Funktionale und Nicht-funktionale Anforderungen

Die funktionalen und nicht-funktionalen Anforderungen beschreiben die erwarteten Eigenschaften und Rahmenbedingungen des Systems. Dabei werden sowohl die konkreten Funktionen der Anwendung als auch Qualitätsmerkmale wie Performance, Benutzerfreundlichkeit und Wartbarkeit betrachtet.

4.1.1 Funktionale Anforderungen

Das entwickelte System soll dem Benutzer ermöglichen, zwei Softwareprojekte miteinander zu vergleichen und Unterschiede übersichtlich darzustellen. Dazu stellt die Anwendung mehrere Funktionen zur Verfügung.

Zunächst muss der Benutzer zwei Softwareprojekte auswählen und hochladen können. Nach erfolgreicher Auswahl wird der Vergleichsvorgang gestartet und der Benutzer wird auf die Vergleichsseite weitergeleitet.

Die Vergleichsseite stellt mehrere Tabs zur Verfügung, über die verschiedene Aspekte des Projektvergleichs angezeigt werden können.

In der Übersichtsseite werden allgemeine Informationen zum Vergleich der beiden Projekte dargestellt. Diese Ansicht bietet dem Benutzer einen schnellen Überblick über die wichtigsten Unterschiede.

Eine zentrale Funktion der Anwendung ist die Darstellung der Projektstruktur in einem Dateibaum. In dieser Ansicht werden Ordner und Dateien beider Projekte hierarchisch angezeigt. Unterschiede zwischen den Projekten werden durch Farben hervorgehoben, um eine schnelle Orientierung zu ermöglichen.

Darüber hinaus unterstützt das System sogenannte FCL-Dateien. Eine FCL-Datei enthält Referenzen auf andere Dateien innerhalb des Projekts. In der Baumstruktur kann optional angezeigt werden, ob eine Datei in einer FCL enthalten ist. Diese Information wird über ein entsprechendes Icon dargestellt und kann ein- oder ausgeblendet werden.

In der Anwendung wird außerdem ein detaillierter Vergleich einzelner Dateien angezeigt. Durch Doppelklick auf eine Datei im Dateibaum öffnet der Benutzer eine Dateiansicht. In dieser Ansicht werden die Unterschiede zwischen den Dateien zeilenweise dargestellt, sodass Änderungen im Code oder in der Konfiguration leicht nachvollzogen werden können.

Zusätzlich stellt das System einen eigenen Tab zur Verfügung, in dem SAP-bezogene Daten angezeigt werden. Diese Ansicht ermöglicht es, relevante SAP-Informationen der beiden Projekte miteinander zu vergleichen.

In der Übersichtsansicht sowie in der SAP-Ansicht werden Vergleichsinformationen tabellarisch dargestellt. In beiden Ansichten kann der Benutzer auswählen, ob nur Unterschiede angezeigt werden sollen oder ob auch identische Inhalte sichtbar bleiben.

4.1.2 Nicht-funktionale Anforderungen

Ein wichtiges Ziel ist eine übersichtliche und benutzerfreundliche Oberfläche. Der Benutzer soll Unterschiede zwischen zwei Softwareprojekten schnell erkennen können. Deshalb werden Änderungen farblich hervorgehoben, sodass neue Dateien oder Ordner grün dargestellt werden und Unterschiede orange markiert sind.

Außerdem muss das System eine ausreichende Performance bieten. Auch bei größeren Softwareprojekten mit vielen Dateien und Ordnern sollen Vergleichsergebnisse in angemessener Zeit angezeigt werden, sodass der Benutzer keine langen Wartezeiten hat.

Ein weiteres wichtiges Kriterium ist die Wartbarkeit und Erweiterbarkeit der Software. Die Anwendung ist daher modular aufgebaut und verwendet Komponenten (siehe Abschnitt 2.1.3). Im Frontend werden Angular-Komponenten und Services (siehe Abschnitt 2.1.4) eingesetzt, während im Backend eine strukturierte Architektur in C# (siehe Abschnitt 2.2.2) verwendet wird, damit das System später leichter angepasst oder erweitert werden kann.

Des Weiteren muss die Anwendung stabil und fehlerrobust arbeiten. Ungültige Eingaben oder unerwartete Zustände sollen abgefangen werden, damit die Anwendung nicht abstürzt und der Benutzer weiterhin mit dem System arbeiten kann.

4.2 Technischer Überblick

Der technische Überblick beschreibt die grundlegende Umsetzung der Anwendung auf Systemebene. Dabei werden die zentralen Komponenten im Frontend und Backend sowie deren technische Struktur und Aufgaben dargestellt.

4.2.1 Frontend

Für die Umsetzung des Frontends wurde das Framework Angular verwendet (siehe Abschnitt 2.2.1). Die Wahl dieser Technologie erfolgte aufgrund der guten Strukturierungsmöglichkeiten sowie der klaren Trennung zwischen Darstellung und Logik.

Das Frontend ist komponentenbasiert aufgebaut. Die Benutzeroberfläche wird durch Angular-Komponenten (siehe Abschnitt 2.1.3) realisiert, welche jeweils für bestimmte Bereiche der Anwendung zuständig sind. Dadurch wird eine modulare Struktur erreicht, die eine einfache Wartung und Erweiterung der Anwendung ermöglicht.

Die Kommunikation mit dem Backend erfolgt über REST-Schnittstellen. Hierfür werden Angular-Services (siehe Abschnitt 2.1.4) eingesetzt, welche die HTTP-Anfragen an das Backend kapseln und die empfangenen Daten den Komponenten zur Verfügung stellen.

Zur Gestaltung der Benutzeroberfläche wird zusätzlich TailwindCSS (siehe Abschnitt 2.4.1) verwendet. Dieses ermöglicht eine flexible und konsistente Gestaltung der Oberfläche und unterstützt die übersichtliche Darstellung der Vergleichsergebnisse.

Die Anwendung wird als Desktop-Applikation mittels Electron (siehe Abschnitt 2.4.1) ausgeführt, wodurch eine plattformunabhängige Nutzung ermöglicht wird. Durch diese Kombination aus Angular und Electron kann die Anwendung als eigenständiges Programm betrieben werden, während gleichzeitig moderne Webtechnologien genutzt werden.

4.2.2 Backend

Für die Umsetzung des Backends wurde die Programmiersprache C# in Kombination mit dem Framework ASP.NET Core verwendet (siehe Abschnitt 2.2.2). Die Wahl dieser Technologie wurde in erster Linie durch die technischen Rahmenbedingungen des Unternehmens vorgegeben.

Innerhalb der Firma ENGEL werden zahlreiche interne Bibliotheken und Pakete in .NET entwickelt und eingesetzt. Durch die Verwendung von C# und ASP.NET konnte das entwickelte System daher problemlos in die bestehende Softwarelandschaft integriert werden. Insbesondere

die firmeneigenen `Engel.Standard`-Bibliotheken (siehe Abschnitt 2.4.2) konnten direkt in das Projekt eingebunden werden.

Ein weiterer Vorteil dieser Technologie liegt in der guten Unterstützung für die Entwicklung von Webschnittstellen. ASP.NET Core ermöglicht eine strukturierte Umsetzung von REST-APIs (siehe Abschnitt 2.1.7). Die klare Trennung zwischen Controllern, Services und Datenmodellen unterstützt dabei eine modulare Architektur und erleichtert sowohl Wartung als auch Erweiterung der Anwendung.

Zusätzlich bietet .NET hohe Stabilität, starke Typisierung sowie eine gute Integration in die Entwicklungsumgebung Visual Studio (siehe Abschnitt 2.3.2). Diese Eigenschaften erleichtern die Entwicklung komplexerer Anwendungen und tragen zur langfristigen Wartbarkeit des Systems bei.

4.2.3 ERP-System

Für den Vergleich bestimmter projektspezifischer Informationen greift das System zusätzlich auf Daten aus dem ERP-System SAP zurück (siehe Abschnitt 2.2.3).

SAP wird im Unternehmen ENGEL als zentrales System zur Verwaltung betrieblicher Daten eingesetzt. In diesem System sind unter anderem projektrelevante Informationen gespeichert, die für die Entwicklung und Konfiguration von Maschinenprojekten benötigt werden.

Im Rahmen der Anwendung werden diese SAP-Daten ausgelesen und zwischen zwei Projekten miteinander verglichen. Dadurch können Unterschiede nicht nur auf Dateiebene, sondern auch auf Ebene der zugehörigen ERP-Daten sichtbar gemacht werden.

Die Integration von SAP ermöglicht somit eine erweiterte Analyse der Projekte, da technische Projektdaten mit den im Unternehmen verwalteten ERP-Informationen kombiniert werden können. Dadurch erhält der Benutzer einen umfassenderen Überblick über mögliche Unterschiede zwischen zwei Softwareprojekten.

4.3 Softwarearchitektur

Die entwickelte Anwendung folgt einer mehrschichtigen Architektur, bei der Frontend, Backend sowie externe Systeme klar voneinander getrennt sind. Diese Struktur ermöglicht eine übersichtliche Organisation der Anwendung und erleichtert sowohl Wartung als auch Erweiterung des Systems.

Die Gesamtarchitektur wurde mit PlantUML grafisch dargestellt, um die wichtigsten Komponenten und deren Kommunikation zu veranschaulichen.

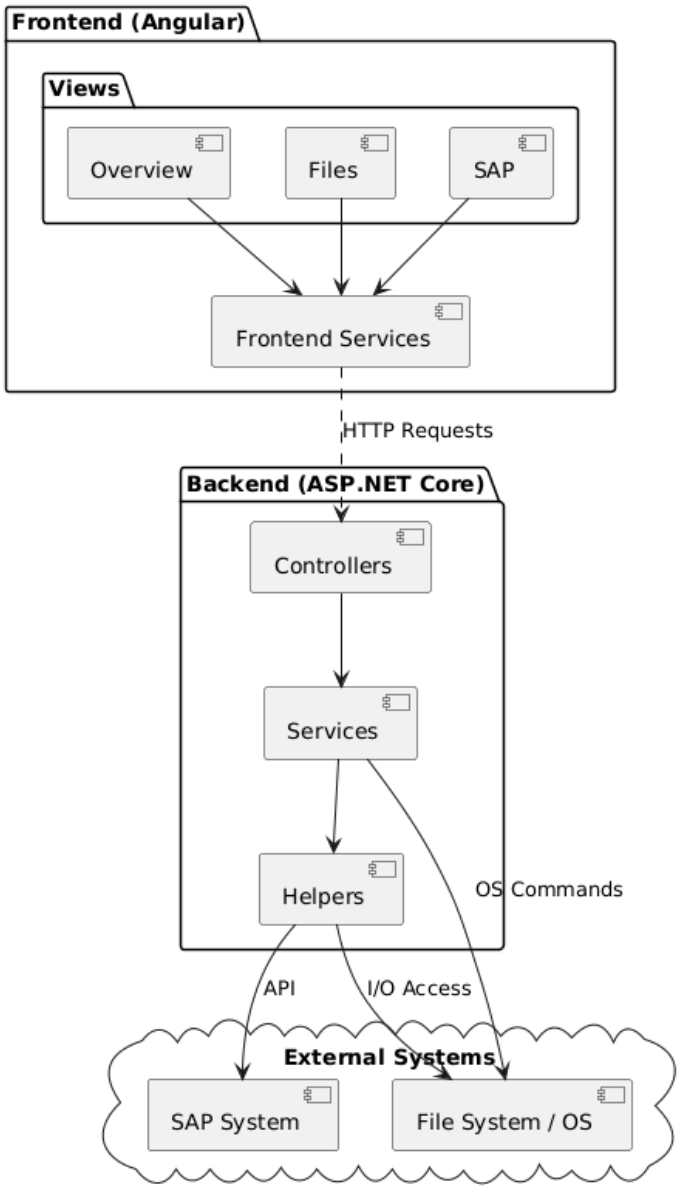


Abbildung 21: Softwarearchitektur

Das **Frontend** ermöglicht dem Benutzer die Auswahl und den Vergleich zweier Softwareprojekte. Die Anwendung besteht aus mehreren Sichten, die jeweils einen bestimmten Vergleichstyp darstellen.

- *Overview* dient zum Vergleich grundlegender Projektinformationen. Die Metadaten werden aus der Datei `ProjectInfo.json` der jeweiligen Projekte ausgelesen und in einer tabellarischen Form gegenübergestellt.
- *Files* dient zum Vergleich der Dateistruktur der beiden Projekte. Dabei werden Unterschiede zwischen Ordnern und Dateien visuell hervorgehoben. Zusätzlich besteht die Möglichkeit, Ordner innerhalb der Benutzeroberfläche zu öffnen sowie detaillierte Änderungen einzelner Dateien anzuzeigen.
- *SAP* dient zum Vergleich projektspezifischer SAP-Daten. Die entsprechenden Informationen werden aus den beiden Projekten ausgelesen und ebenfalls in einer strukturierten Form gegenübergestellt.

Innerhalb des Frontends werden verschiedene Services verwendet, die die Kommunikation mit dem Backend sowie interne Hilfsfunktionen übernehmen. Diese Services kapseln die Logik für API-Aufrufe und stellen die benötigten Daten den einzelnen Sichten zur Verfügung.

Das **Backend** übernimmt die Verarbeitung der Anfragen des Frontends und stellt die zentralen Funktionen für den Projektvergleich bereit. Es stellt mehrere REST-Endpunkte bereit, über die die verschiedenen Vergleichsoperationen ausgelöst werden können. Die Controller fungieren dabei als Schnittstelle zwischen Frontend und Backendlogik. Sie empfangen HTTP-Anfragen und leiten diese an die entsprechenden Services weiter. Die eigentliche Verarbeitung findet innerhalb der Service-Schicht statt. Dort werden die übergebenen Projektpfade analysiert und die notwendigen Vergleichsoperationen durchgeführt. Dazu gehört beispielsweise das Einlesen von Projektinformationen, die Analyse der Dateistruktur sowie der Vergleich einzelner Dateien. Zusätzlich werden Hilfskomponenten verwendet, die spezielle Aufgaben wie Dateizugriffe, Logging oder weitere technische Funktionen übernehmen.

Für bestimmte Funktionen greift das System auf **externe Systeme** zu. Dazu zählt insbesondere das SAP-System des Unternehmens, aus dem projektrelevante Daten abgefragt werden können. Darüber hinaus erfolgt der Zugriff auf die eigentlichen Softwareprojekte über das Dateisystem des Betriebssystems. Die Pfade der zu vergleichenden Projekte werden vom Frontend übergeben und anschließend im Backend analysiert.

4.4 Projektstruktur

Die Projektstruktur beschreibt den Aufbau und die Organisation des Quellcodes innerhalb der Anwendung. Dabei werden die wichtigsten Verzeichnisse und deren Aufgaben im Frontend und Backend dargestellt.

4.4.1 Frontend

Die Struktur des Frontends ist in mehrere logisch getrennte Bereiche unterteilt. Die zentrale Implementierung befindet sich im Verzeichnis `src/app`, in dem die Komponenten der Benutzeroberfläche, die zugehörigen Services sowie gemeinsame Datenstrukturen organisiert sind.

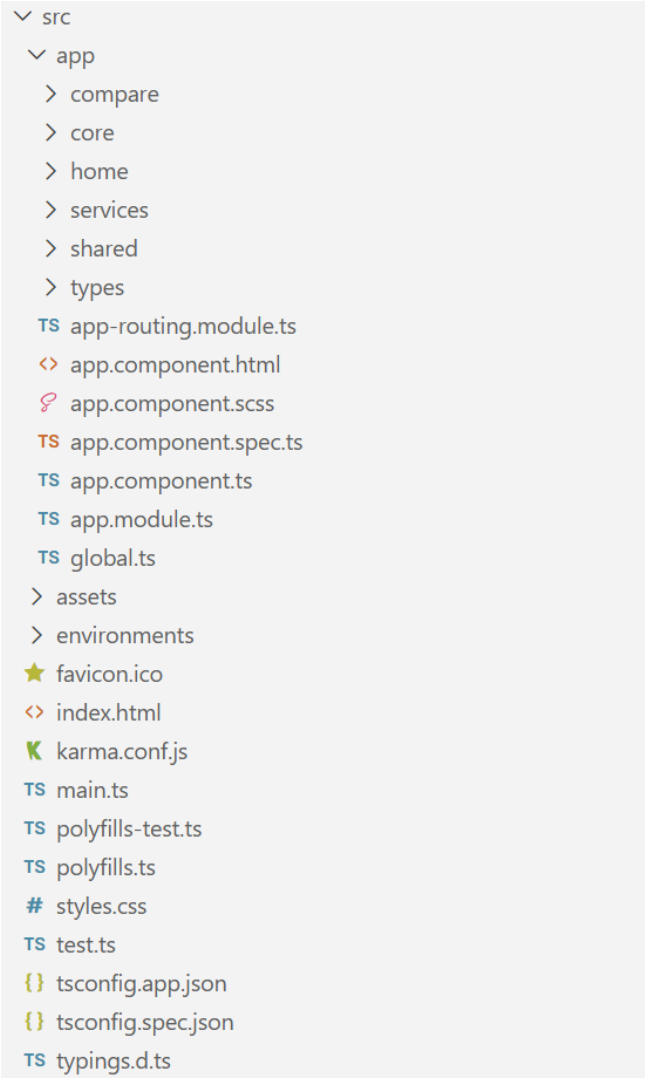


Abbildung 22: Projektstruktur Frontend

Abbildung 22 zeigt die grundlegende Struktur des Frontend-Projekts.

Die wichtigsten Bestandteile werden im Folgenden erläutert.

Compare

Der Ordner `compare` enthält Angular-Komponenten (siehe Abschnitt 2.1.3) für den eigentlichen Vergleich der beiden Softwareprojekte. Hier befinden sich die Sichten für die verschiedenen Vergleichsarten, beispielsweise der Dateivergleich, die Übersicht der Unterschiede sowie die Darstellung einzelner Dateien.

Diese Komponenten bilden den wesentlichen Funktionsbereich der Anwendung, da hier die vom Backend gelieferten Vergleichsdaten visualisiert werden.

Home

Der Ordner `home` enthält die Startseite der Anwendung. Auf dieser Seite kann der Benutzer die beiden zu vergleichenden Softwareprojekte auswählen und den Vergleichsvorgang starten.

Nach der Auswahl der Projekte wird der Benutzer auf die Vergleichsansicht weitergeleitet.

Core

Der Ordner `core` enthält zentrale Funktionen der Anwendung, die global benötigt werden. Dazu gehört insbesondere der `ElectronService`, der die Kommunikation zwischen der Angular-Anwendung (siehe Abschnitt 2.2.1) und der Electron-Umgebung (siehe Abschnitt 2.4.1) ermöglicht.

Services

Der Ordner `services` enthält Angular-Services (siehe Abschnitt 2.1.4), die für die Kommunikation mit dem Backend zuständig sind. Die Services kapseln die HTTP-Anfragen an die REST-API des Backends und stellen die empfangenen Daten den Frontend-Komponenten zur Verfügung.

Shared

Der Ordner `shared` enthält wiederverwendbare Komponenten (siehe Abschnitt 2.1.3), die in mehreren Teilen der Anwendung eingesetzt werden können.

Durch die zentrale Verwaltung dieser Elemente wird redundanter Code vermieden und die Wiederverwendbarkeit der Komponenten verbessert.

Types

Im Ordner `types` befinden sich TypeScript-Typdefinitionen und Interfaces, die die Struktur der Daten beschreiben, welche zwischen Frontend und Backend ausgetauscht werden.

Diese Typen definieren beispielsweise die Struktur eines Dateibaums, eines Vergleichsergebnisses oder der Projektinformationen.

Routing und Hauptmodule

Die Datei `app-routing.module.ts` definiert das Routing der Anwendung. Hier wird festgelegt, welche Komponente (siehe Abschnitt 2.1.3) für eine bestimmte URL geladen wird.

Das zentrale Angular-Modul der Anwendung ist `app.module.ts`. In dieser Datei werden alle Komponenten, Module und Services der Anwendung registriert und miteinander verbunden.

Weitere Projektdateien

Neben dem Ordner `src/app` enthält das Projekt weitere Konfigurations- und Builddateien, beispielsweise `angular.json`, `package.json` oder `tsconfig.json`. Diese Dateien werden hauptsächlich für den Build-Prozess, die Abhängigkeitsverwaltung sowie die Konfiguration der Entwicklungsumgebung verwendet und enthalten keine direkte Anwendungslogik.

4.4.2 Backend

Um eine klare Trennung der Zuständigkeiten (Separation of Concerns [7]) sowie eine gute Wartbarkeit und Erweiterbarkeit der Anwendung zu gewährleisten, wurde das Backend in verschiedene Ordner und Namespaces aufgeteilt.

Abbildung 23 zeigt die grundlegende Verzeichnisstruktur von *ProjectCompare*.

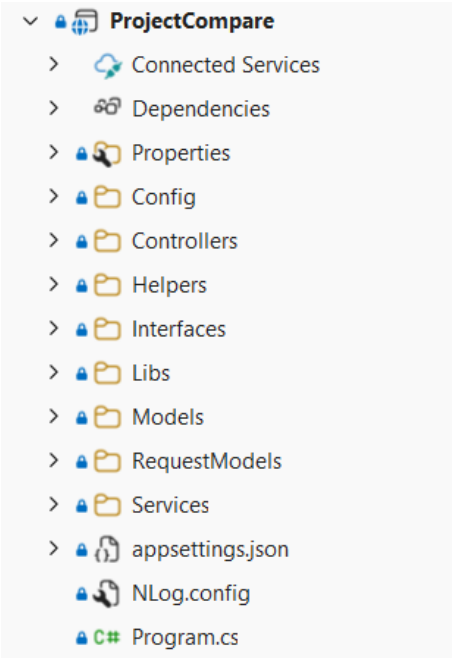


Abbildung 23: Projektstruktur Backend

Die einzelnen Ordner übernehmen jeweils klar definierte Aufgaben innerhalb des Systems. Im Folgenden werden die wichtigsten Bestandteile und deren Zusammenspiel näher erläutert.

Controllers

Der Ordner `Controllers` enthält die API-Controller, die als Schnittstelle zwischen Backend und Angular-Frontend dienen. Sie nehmen eingehende HTTP-Requests entgegen – beispielsweise zwei Pfade zu den zu vergleichenden Projekten – und übergeben diese an die entsprechenden Services zur weiteren Verarbeitung.

Die Controller sind damit hauptsächlich für Routing, Validierung der Eingabedaten sowie die Übergabe der Parameter zuständig, enthalten jedoch keine umfangreiche Geschäftslogik. Entsprechend den Tabs im Frontend existieren hier Endpunkte für den `OverviewCompare`, `FileCompare` und `SapCompare`.

Services

Die zentrale Geschäftslogik der Anwendung befindet sich im Ordner `Services`. Hier werden die eigentlichen Vergleichsoperationen durchgeführt.

Beispielsweise übernimmt ein Service (siehe Abschnitt 2.1.4) das Einlesen und Vergleichen der Datei `ProjectInfo.json` für die Overview-Ansicht. Ein weiterer Service analysiert rekursiv die Ordnerstruktur der Projekte, um Unterschiede, Änderungen oder fehlende Dateien für die Baumansicht im File-Vergleich zu ermitteln. Zusätzlich existiert ein Service, der die relevanten SAP-Daten zusammenführt und miteinander vergleicht.

Interfaces

Um die Komponenten möglichst lose zu koppeln und die Dependency Injection zu nutzen, werden die Schnittstellen der Services im Ordner `Interfaces` definiert (z. B. `IFileCompareService`).

Die Controller arbeiten ausschließlich mit diesen Interfaces (siehe Abschnitt 2.1.8) und nicht direkt mit den konkreten Implementierungen. Dadurch wird die Modularität der Anwendung erhöht und das automatisierte Testen erleichtert, da Services bei Bedarf durch Mock-Implementierungen ersetzt werden können.

Models und RequestModels

Models: In diesem Ordner befinden sich die Objekt-Klassen und DTOs, die innerhalb des Backends sowie für die Antworten an das Frontend verwendet werden. Diese Modelle repräsentieren beispielsweise die Knoten eines Dateibaums (inklusive Eigenschaften wie Dateiname oder Änderungsstatus) oder die Struktur der eingelesenen Konfigurationsdateien.

RequestModels: Die Klassen in diesem Ordner definieren die Struktur der Daten, die vom Angular-Frontend an das Backend übermittelt werden. Eine typische Request-Klasse enthält beispielsweise zwei Strings für die Pfade von „Projekt A“ und „Projekt B“. Die Trennung zwischen Request- und Response-Modellen verbessert die Typsicherheit und sorgt für eine klar definierte Datenstruktur in der Kommunikation zwischen Frontend und Backend.

Helpers

Der Ordner `Helpers` enthält Hilfsklassen. Dabei handelt es sich meist um Funktionen, die an mehreren Stellen der Anwendung benötigt werden, beispielsweise für Konvertierungen, die Verarbeitung von Dateipfaden oder das Einlesen von Dateiinhalten.

Config

Im Ordner `Config` befinden sich Klassen zur Verwaltung spezifischer Konfigurationen für den Projektvergleich. Im Gegensatz zur allgemeinen Serverkonfiguration, zum Beispiel in der `appsettings.json`, werden hier anwendungsspezifische Einstellungen für den Projektvergleich definiert und den entsprechenden Services zur Verfügung gestellt.

Properties

Der Ordner `Properties` enthält Konfigurationsdateien, die das Startverhalten der Anwendung während der Entwicklung festlegen und in der Regel keinen Einfluss auf die produktive Anwendung haben.

Die wichtigste Datei ist `launchSettings.json`. Darin wird definiert, wie das Backend lokal gestartet wird, zum Beispiel über welche Ports der Webserver erreichbar ist und welche Umgebungsvariablen gesetzt werden. Außerdem können unterschiedliche Startprofile (z. B. für HTTP oder IIS Express) hinterlegt werden.

Libs, Connected Services und Dependencies

Libs / Dependencies: Dieser Bereich umfasst externe Bibliotheken und NuGet-Pakete, die im Projekt verwendet werden, beispielsweise für den textuellen Vergleich von Dateien (siehe Abschnitt 2.4.2).

Connected Services: Falls externe Dienste oder APIs angebunden sind, wird die automatisch generierte Client-Infrastruktur in diesem Bereich verwaltet. Diese wurden aber in unserem Projekt nicht verwendet.

Konfigurations- und Startdateien im Root-Verzeichnis

`Program.cs`: Diese Datei ist der Einstiegspunkt der Anwendung. Hier wird das Backend gestartet und grundlegende Einstellungen werden vorgenommen. Dazu gehört unter anderem die

Konfiguration des Webservers, das Einrichten der Middleware (z. B. CORS-Zugriff) sowie die Registrierung der benötigten Services.

`appsettings.json`: Diese JSON-Datei enthält grundlegende Konfigurationen der Anwendung. Dazu gehören unter anderem Einstellungen für das Logging sowie allgemeine Serveroptionen. In der aktuellen Implementierung wird hier beispielsweise festgelegt, welche Log-Level verwendet werden und welche Hosts auf die Anwendung zugreifen dürfen.

`NLog.config`: Diese Datei enthält die Konfiguration des Logging-Frameworks *NLog* (siehe Abschnitt 2.4.2). Sie definiert, wie und wohin Logeinträge wie Fehler, Warnungen oder Debug-Informationen geschrieben werden. Dies ist insbesondere für die Analyse von Fehlern während der Vergleichsprozesse wichtig.

5 Implementierung

Die Anwendung wurde als Fullstack-System mit klarer Trennung von Frontend und Backend umgesetzt. Die Kommunikation erfolgt über eine REST-Schnittstelle, wodurch eine standardisierte Datenübertragung ermöglicht wird.

5.1 Webanwendung

Das Frontend wurde als Webanwendung mit Angular umgesetzt und ermöglicht dem Benutzer, zwei Projekte auszuwählen, den Vergleich zu starten und die Ergebnisse in verschiedenen Darstellungsformen zu analysieren. Die Oberfläche ist komponentenbasiert aufgebaut, wodurch einzelne Teile der Anwendung klar strukturiert und wiederverwendbar sind. Zusätzlich sorgen visuelle Hervorhebungen und verschiedene Ansichten, wie Baumstruktur oder Detailansichten, für eine übersichtliche Darstellung der Unterschiede.

5.1.1 Typen

Die verwendeten Typen beschreiben den Aufbau der verarbeiteten Daten. Dazu gehören sowohl die Antworten des Backends als auch interne Zustände der Anwendung.

TableDifference

Der Typ `TableDifference` beschreibt einen einzelnen Vergleichseintrag in einer tabellarischen Ansicht. Er wird beispielsweise für die Übersichtsdaten und die SAP-Vergleichsdaten verwendet.

```
1  export interface TableDifference {
2      name: string;
3      project1Value: string;
4      project2Value: string;
5      isDifferent: boolean;
6  }
```

Das Feld `name` enthält die Bezeichnung des verglichenen Eintrags. Die Felder `project1Value` und `project2Value` enthalten die jeweiligen Werte der beiden Projekte. Über `isDifferent` wird gespeichert, ob sich die beiden Werte unterscheiden.

CompareProjectsResponse

Der Typ `CompareProjectsResponse` beschreibt eine zusammengefasste Antwort für einen Projektvergleich. Er bündelt die Übersichtsdaten und die SAP-Daten in einer gemeinsamen Struktur.

```
export interface CompareProjectsResponse {
    overview: TableDifference[];
    sap: TableDifference[];
}
```

Das Feld `overview` enthält die allgemeinen Vergleichsdaten der beiden Projekte. Das Feld `sap` enthält die Vergleichsdaten mit SAP-Bezug. Beide Eigenschaften bestehen jeweils aus mehreren `TableDifference`-Einträgen.

FileNodeStatus

Der Enum `FileNodeStatus` beschreibt den Vergleichsstatus eines Dateiknotens innerhalb der Projektstruktur.

```
export enum FileNodeStatus {
    Only = 'Only',
    Missing = 'Missing',
    Different = 'Different',
    Same = 'Same',
}
```

Der Wert `Only` kennzeichnet Einträge, die nur in einem Projekt vorhanden sind. `Missing` steht für fehlende Einträge, `Different` für unterschiedliche Inhalte und `Same` für identische Dateien oder Ordner.

FileNodeDifference

Der Typ `FileNodeDifference` beschreibt einen Knoten in der Dateibaumstruktur eines Projekts. Er wird verwendet, um Dateien und Ordner mitsamt ihrem Vergleichsstatus im Frontend darzustellen.

```
export interface FileNodeDifference {
    name: string;
    isDirectory: boolean;
    inFcl: boolean;
    relativePath: string;
    status: FileNodeStatus;
    children?: FileNodeDifference[];
}
```

Das Feld `name` enthält den Namen des Knotens. Über `isDirectory` wird festgehalten, ob es sich um einen Ordner oder eine Datei handelt. Das Feld `relativePath` speichert den relativen Pfad innerhalb des Projekts. Der Status wird über `FileNodeStatus` beschrieben. Mit `children` kann die hierarchische Baumstruktur rekursiv aufgebaut werden.

FileStructureComparison

Der Typ `FileStructureComparison` beschreibt die vollständige Dateistruktur zweier Projekte für den Vergleich im Baum.

```
export interface FileStructureComparison {
    project1: FileNodeDifference[];
    project2: FileNodeDifference[];
    project1Path: string;
    project2Path: string;
}
```

Die Felder `project1` und `project2` enthalten jeweils die Dateibaumstruktur eines Projekts. Zusätzlich speichern `project1Path` und `project2Path` die absoluten Pfade der verglichenen Projekte.

TreeViewState

Der Typ `TreeViewState` beschreibt den gespeicherten Zustand einer Baumansicht im Frontend.

```
export interface TreeViewState {
    expandedNodeIndices: number[];
    scrollY: number;
}
```

Das Feld `expandedNodeIndices` speichert die Positionen der aktuell geöffneten Knoten innerhalb des Baums. Das Feld `scrollY` enthält die vertikale Scroll-Position. Dadurch kann die Baumansicht später in demselben Zustand wiederhergestellt werden.

DiffLineStatus

Der Enum `DiffLineStatus` beschreibt den Vergleichsstatus einer einzelnen Zeile in einem Dateivergleich.

```
export enum DiffLineStatus {
    Only = 'Only',
    Missing = 'Missing',
    Different = 'Different',
    Same = 'Same'
}
```

Die Werte entsprechen den möglichen Zuständen einer Zeile im Vergleich. Damit kann in der Diff-Ansicht unterschieden werden, ob eine Zeile nur auf einer Seite vorkommt, fehlt, verschieden ist oder identisch vorliegt.

DiffContent

Der Typ `DiffContent` beschreibt einen einzelnen Inhaltsteil innerhalb einer Zeile. Er wird verwendet, um Unterschiede nicht nur zeilenweise, sondern auch innerhalb einer Zeile hervorzuheben.

```
export interface DiffContent {
    content: string,
    isMarked: boolean
}
```

Das Feld `content` enthält den eigentlichen Textabschnitt. Mit `isMarked` wird festgelegt, ob dieser Abschnitt in der Oberfläche hervorgehoben dargestellt werden soll.

DiffLine

Der Typ `DiffLine` beschreibt eine einzelne Zeile innerhalb eines Dateivergleichs. Er kombiniert Zeilennummer, Inhalt und Vergleichsstatus.

```
export interface DiffLine {
    lineNumber: string,
    content: DiffContent[],
    status: DiffLineStatus
}
```

Das Feld `lineNumber` enthält die Zeilennummer der Datei. `content` besteht aus mehreren `DiffContent`-Elementen, damit auch Teilbereiche einer Zeile markiert werden können. Der Status der gesamten Zeile wird über `DiffLineStatus` beschrieben.

FileComparison

Der Typ `FileComparison` beschreibt das Ergebnis eines Vergleichs zwischen zwei Dateien. Er wird vom Backend an das Frontend geliefert und anschließend in der Diff-Ansicht dargestellt.

```
export interface FileComparison {
    file1Lines: DiffLine[],
    file2Lines: DiffLine[]
}
```

Die Eigenschaft `file1Lines` enthält die Zeilen der ersten Datei, `file2Lines` die Zeilen der zweiten Datei. Beide Seiten bestehen jeweils aus einer Liste von `DiffLine`-Objekten, wodurch Unterschiede strukturiert in der Oberfläche visualisiert werden können.

5.1.2 Services

Es werden Angular-Services eingesetzt (siehe Abschnitt 2.1.4), um wiederverwendbare Logik zu kapseln und die Kommunikation mit dem Backend zu realisieren.

FileCompareService

Der `FileCompareService` wird verwendet, um zwei Dateien miteinander zu vergleichen. Dazu sendet der Service die Dateipfade an das Backend und erhält die berechneten Vergleichsdaten zurück. Diese Daten werden anschließend in der Diff-Ansicht der Anwendung dargestellt.

```
1  public getFileComparison(filePath1: string, filePath2: string): Observable<FileComparison>{
2      const params = {
3        filePath1: filePath1,
4        filePath2: filePath2
5      };
6
7      return this.http.get<FileComparison>(`${endpoint}/Compare/Files`, { params });
8  }
```

Die Methode `getFileComparison` erhält die Dateipfade der beiden zu vergleichenden Dateien als Parameter. Diese werden als Request-Parameter an den Backend-Endpunkt `/Compare/Files` übergeben.

Der Aufruf erfolgt über den Angular `HttpClient`. Die Methode gibt ein `Observable<FileComparison>` zurück, das die Vergleichsdaten der beiden Dateien enthält.

ProjectCompareService

Der `ProjectCompareService` ist für den Vergleich auf Projektebene zuständig. Er lädt die zentralen Vergleichsdaten zweier Projekte aus dem Backend und stellt diese der Oberfläche zur Verfügung. Dazu gehören die Übersichtsdaten, SAP-bezogene Vergleichsdaten sowie die Dateistruktur. Zusätzlich werden die Projektnamen aus den geladenen Daten übernommen, damit sie in der UI direkt angezeigt werden können.

```
public loadCompareData(project1Path: string, project2Path: string): Observable<void> {
    const params = {
      projectPath1: project1Path,
      projectPath2: project2Path
    };

    const overviewData = this.http.get<any>(`${endpoint}/Compare/Projects/Overview`, {
        params }).pipe(
      map(raw => {
        const parsed = parseOverviewResponse(raw);
        const line = parsed.find(item => item.name === 'Machine.ProjectName');
        this.project1Name.next(line?.project1Value || 'Project 1');
        this.project2Name.next(line?.project2Value || 'Project 2');
        this.overviewDataSubject.next(parsed);
      })
    );

    const sapData = this.http.get<any>(`${endpoint}/Compare/Projects/Sap`, { params
        }).pipe(
      map(raw => {
        const parsed = parseSAPResponse(raw);
        this.sapDataSubject.next(parsed);
      })
    );

    const filesData = this.http.get<any>(`${endpoint}/Compare/Projects/FileStructure`, {
        params }).pipe(
      map(raw => {
        const parsed = parseFileStructureResponse(raw);
        this.fileStructureSubject.next(parsed);
      })
    );

    return forkJoin([overviewData, sapData, filesData]).pipe(map(() => void 0));
}
```

Die Methode `loadCompareData` erhält die beiden Projektpfade und ruft damit drei Backend-Endpunkte auf. Die Antworten werden in das jeweilige Datenmodell umgewandelt und anschließend in den internen Datenströmen gespeichert. Dabei verwenden die Übersichts- und SAP-Daten den Typ `TableDifference[]`, während die Dateistruktur den Typ `FileStructureComparison` verwendet. Durch `forkJoin` wird sichergestellt, dass der gesamte Ladevorgang erst dann abgeschlossen ist, wenn alle drei Requests erfolgreich beendet wurden.

```
public clearData(): void {
    this.overviewDataSubject.next([]);
    this.sapDataSubject.next([]);
    this.fileStructureSubject.next({ project1: [], project2: [], project1Path: '',
        project2Path: '' });
}
```

Die Methode `clearData` setzt die zuvor geladenen Vergleichsdaten auf einen leeren Zustand zurück. Dabei werden die Datenströme für die Übersicht und die SAP-Daten auf leere Arrays des Typs `TableDifference[]` gesetzt. Die Dateistruktur wird auf ein leeres Objekt des Typs `FileStructureComparison` zurückgesetzt.

SelectedFileService

Der `SelectedFileService` wird verwendet, um die aktuell ausgewählte Datei zentral im Frontend zu speichern. Dadurch können mehrere Komponenten auf denselben Auswahlzustand zugreifen, ohne direkt miteinander gekoppelt zu sein. Wenn der Benutzer im Dateibaum eine

andere Datei auswählt, wird dieser Zustand im Service aktualisiert und kann von anderen Teilen der Anwendung weiterverwendet werden.

```
setSelectedFile(file: FileNodeDifference | null): void {
    this.selectedFileSubject.next(file);
}
```

Die Methode `setSelectedFile` übernimmt die aktuell ausgewählte Datei als Parameter und speichert diesen Wert im internen `BehaviorSubject`. Der übergebene Wert verwendet dabei den Typ `FileNodeDifference`.

```
clearSelectedFile(): void {
    this.selectedFileSubject.next(null);
}
```

Die Methode `clearSelectedFile` setzt die aktuelle Auswahl auf `null`. Dadurch wird signalisiert, dass keine Datei mehr ausgewählt ist. Diese Methode wird beispielsweise verwendet, wenn die Ansicht zurückgesetzt wird oder ein neuer Vergleich gestartet wird.

Der Zustand wird über ein Observable (siehe Abschnitt 2.4.1) bereitgestellt, sodass Komponenten bei Änderungen automatisch reagieren können, ohne die Daten aktiv abfragen zu müssen.

TreeViewStateService

Der `TreeViewStateService` wird verwendet, um den aktuellen Zustand der Baumansicht im Frontend zu speichern. Dazu gehören insbesondere die geöffneten Knoten sowie die aktuelle Scroll-Position. Dadurch kann die Anwendung beim Wechsel zwischen Ansichten oder beim erneuten Laden der Baumstruktur den vorherigen Zustand wiederherstellen und die Navigation für den Benutzer konsistent halten.

```
saveState(
    key: string,
    treeControl: NestedTreeControl<FileNodeDifference>,
    dataSource: MatTreeNestedDataSource<FileNodeDifference>,
    scrollY: number
): void {
    const expandedNodeIndices: number[] = [];

    const getAllNodes = (nodes: FileNodeDifference[], currentIndex = { value: 0 }): void
        => {
      nodes.forEach(node => {
        if (treeControl.isExpanded(node)) {
          expandedNodeIndices.push(currentIndex.value);
        }
        currentIndex.value++;

        if (node.children && node.children.length > 0) {
          getAllNodes(node.children, currentIndex);
        }
      });
    };

    getAllNodes(dataSource.data);

    this.stateMap.set(key, {
      expandedNodeIndices,
      scrollY
    });
}
```

Die Methode `saveState` speichert den Zustand einer Baumansicht unter einem eindeutigen Schlüssel. Dazu wird rekursiv durch alle Knoten der Datenstruktur iteriert und für jeden geöffneten Knoten dessen Position gespeichert. Die Knoten der Baumstruktur verwenden dabei den Typ `FileNodeDifference`. Zusätzlich wird die aktuelle vertikale Scroll-Position übernommen. Auf diese Weise kann der Zustand der Baumansicht zu einem späteren Zeitpunkt wiederhergestellt werden.

```
1  hasState(key: string): boolean {
2      return this.stateMap.has(key);
3  }
```

Die Methode `hasState` prüft, ob für einen bestimmten Schlüssel bereits ein Zustand gespeichert wurde. Dadurch kann vor dem Wiederherstellen der Baumansicht kontrolliert werden, ob überhaupt Daten vorhanden sind.

```
1  getState(key: string): TreeViewState | undefined {
2      return this.stateMap.get(key);
3  }
```

Die Methode `getState` liefert den zuvor gespeicherten Zustand zu einem bestimmten Schlüssel zurück. Falls für diesen Schlüssel kein Zustand vorhanden ist, gibt die Methode `undefined` zurück.

```
1  clearState(key: string): void {
2      this.stateMap.delete(key);
3  }
```

Die Methode `clearState` entfernt den gespeicherten Zustand einer bestimmten Baumansicht. Dies ist dann sinnvoll, wenn ein Zustand nicht mehr benötigt wird oder bewusst zurückgesetzt werden soll.

```
1  clearAllStates(): void {
2      this.stateMap.clear();
3  }
```

Die Methode `clearAllStates` löscht alle gespeicherten Zustände aus dem Service. Sie kann verwendet werden, wenn die Anwendung vollständig in einen neuen Zustand versetzt werden soll, zum Beispiel beim Start eines neuen Vergleichs.

UtilityService

Der `UtilityService` stellt verschiedene technische Hilfsfunktionen für das Frontend bereit. Er wird für allgemeine Aufgaben verwendet, die nicht direkt zu einer bestimmten Fachfunktion der Anwendung gehören.

```
1  public openFolder(folderPath: string): void {
2      const body = { folderPath: folderPath };
3      this.http.post(${endpoint}/Utility/OpenFolder, body, { responseType:
           'text'}).subscribe();
4  }
```

Die Methode `openFolder` wird verwendet, um einen Ordner im Dateisystem zu öffnen. Dazu wird der übergebene Ordnerpfad an den Backend-Endpunkt `/Utility/OpenFolder` gesendet.

Der Aufruf erfolgt über einen HTTP-POST-Request mit dem Angular `HttpClient`. Das Backend verarbeitet den Request und öffnet anschließend den entsprechenden Ordner im Betriebssystem. Da lediglich die Aktion ausgeführt werden soll, wird die Antwort des Requests nicht weiter verarbeitet und der Aufruf direkt über `subscribe()` gestartet.

5.1.3 Komponenten

Wie im Abschnitt 2.1.3 erwähnt, bilden Komponenten die Darstellungsschicht der Webanwendung. Sie übernehmen die Benutzerinteraktion, zeigen die Vergleichsdaten an und binden die Services (siehe Abschnitt 2.1.4) in die Oberfläche ein.

Den Einstiegspunkt bildet die AppComponent, welche die Root-Komponente ist und das Routing-Layout der Anwendung einbindet. Eine eigene Fachlogik ist in dieser Komponente nicht enthalten.

HomeComponent

Die HomeComponent dient als Startseite und übernimmt die Auswahl der beiden Projektpfade. Nach erfolgreicher Auswahl wird zur Vergleichsansicht navigiert.

```
1  public onPathSelected(projectNumber: number, path: string): void {
2    if (projectNumber === 1) {
3      this.filePathProject1 = path;
4    } else if (projectNumber === 2) {
5      this.filePathProject2 = path;
6    }
7  }
8
9  public navigateToCompare(): void {
10   if (!this.filePathProject1 || !this.filePathProject2) return;
11
12   this.projectCompareService.clearData();
13
14   this.router.navigate(['/compare'], {
15     queryParams: {
16       project1: this.filePathProject1,
17       project2: this.filePathProject2
18     },
19   });
20 }
```

Die Methode `onPathSelected` speichert die Pfade aus den Upload-Komponenten. Die Methode `navigateToCompare` prüft die Eingaben, setzt alte Vergleichsdaten zurück und startet die Navigation zur Compare-Seite.

FolderUploadComponent

Die FolderUploadComponent kapselt die Ordnerauswahl über Electron und gibt den gewählten Pfad an die Parent-Komponente weiter.

```
async selectFolder() {
  const folderPath = await ipcRenderer.invoke('select-folder');
  this.path = folderPath;
  this.pathSelected.emit(this.path);
}
```

Die Methode `selectFolder` öffnet den nativen Ordnerdialog, übernimmt den ausgewählten Pfad und sendet ihn per EventEmitter an die HomeComponent.

CompareComponent

Die CompareComponent ist der Container der Vergleichsseite. Sie lädt die Daten initial und steuert den Wechsel zwischen Overview-, Files- und SAP-Tab.

```
ngOnInit(): void {
  this.projectCompareService.loadCompareData(this.project1, this.project2).subscribe();
}

onTabClick(tab: 'overview' | 'files' | 'sap'): void {
  this.activeTab = tab;
  this.treeViewStateService.clearAllStates();
  this.selectedFileService.clearSelectedFile();
  this.showSettings = false;
}
```

`ngOnInit` startet den Projektvergleich beim Laden der Seite. `onTabClick` wechselt den Tab und setzt zustandsabhängige UI-Daten zurück.

OverviewTabComponent

Die OverviewTabComponent zeigt die allgemeinen Vergleichsdaten in Tabellenform an. Die Komponente übernimmt vor allem Darstellung und bezieht die Daten über den ProjectCompareService.

SapTabComponent

Die SapTabComponent zeigt die SAP-spezifischen Vergleichsdaten. Wie beim Overview-Tab werden die Vergleichsdaten in Tabellenform angezeigt und die Daten werden vom Service bezogen.

FilesTabComponent

Die FilesTabComponent koordiniert den Datei-Vergleichsbereich. Sie verbindet Dateibaum, Dateiauswahl und Detailvergleich.

```
ngOnInit(): void {
  this.subscriptions.add(
    this.projectCompareService.fileStructureData$.subscribe((data) => {
      this.fileDataProject1 = data.project1;
      this.fileDataProject2 = data.project2;
      this.project1Path = data.project1Path;
      this.project2Path = data.project2Path;
    })
  );

  this.subscriptions.add(
    this.selectedFileService.selectedFile$.subscribe(file => {
      this.selectedFile = file;
      this.showFilesDifference = file !== null && file !== undefined;
    })
  );
}
```

Die Methode `ngOnInit` verbindet die Komponente mit den zentralen Datenströmen für Dateistruktur und Dateiauswahl. Dadurch wird die Ansicht automatisch aktualisiert, sobald sich die Daten ändern.

TreeCompareComponent

Die TreeCompareComponent stellt zwei Baumansichten nebeneinander dar und synchronisiert deren Verhalten.

```
syncScroll(source: 'left' | 'right'): void {
  if (this.isSyncingScroll) return;
  this.isSyncingScroll = true;

  const leftScrollElement = this.leftTreeComponent?.panelRef?.nativeElement;
  const rightScrollElement = this.rightTreeComponent?.panelRef?.nativeElement;

  if (source === 'left') {
    rightScrollElement.scrollTop = leftScrollElement.scrollTop;
  } else {
    leftScrollElement.scrollTop = rightScrollElement.scrollTop;
  }

  setTimeout(() => this.isSyncingScroll = false, 10);
}
```

`syncScroll` stellt sicher, dass beide Baumansichten beim Scrollen synchron bleiben und verhindert gleichzeitig Endlosschleifen durch reziproke Scroll-Events.

TreeViewComponent

Die TreeViewComponent zeigt eine einzelne Dateibaumstruktur und verarbeitet Knoteninteraktionen.

```
public onClick(node: FileNodeDifference): void {
  this.saveTreeState();
  this.selectedFileService.setSelectedFile(node);
}

public onToggleNode(node: FileNodeDifference): void {
  this.treeControl.toggle(node);
  const index = this.findNodeIndex(node);
  const isNowExpanded = this.treeControl.isExpanded(node);
  if (!this.isInternalToggle) {
    this.nodeToggled.emit({ node, isExpanded: isNowExpanded, index });
  }
}
```

`onClick` speichert zuerst den aktuellen Baumzustand und setzt dann die ausgewählte Datei. `onToggleNode` erweitert oder reduziert Knoten und informiert die zweite Baumseite über den neuen Zustand.

FileCompareComponent

Die FileCompareComponent zeigt den zeilenweisen Vergleich zweier Dateien und lädt die Diff-Daten.

```
loadFileComparison(): void {
  const filePath1 = `${this.absolutePathProject1}${this.file.relativePath}`;
  const filePath2 = `${this.absolutePathProject2}${this.file.relativePath}`;

  this.fileCompareService.getFileComparison(filePath1, filePath2).subscribe({
    next: (data: FileComparison) => {
      this.selectedFileComparison = data;
    },
    error: err => {
      console.error('Fehler beim Laden der Vergleichsdaten:', err);
    }
  });
}
```

Die Methode erzeugt aus Projektpfad und Relativpfad die beiden absoluten Dateipfade und ruft anschließend den FileCompareService auf. Das Ergebnis wird in der Diff-Ansicht angezeigt.

CompareTableComponent

Die CompareTableComponent ist eine wiederverwendbare Tabellenkomponente für Overview- und SAP-Daten.

```
public filteredData(): TableDifference[] {
  if (this.showDifferencesOnly) {
    return this.overviewData.filter(row => row.isDifferent);
  }
  return this.overviewData;
}
```

Die Methode `filteredData` steuert, ob alle Einträge oder nur Unterschiede angezeigt werden.

FileViewComponent

Die FileViewComponent rendert die Diff-Zeilen einer Seite und liefert die farbliche Hervorhebung pro Zeilenstatus.

```
getLineClass(status: string): string {
  switch (status) {
    case 'Same': return 'text-black';
    case 'Different': return 'bg-orange-300/20';
    case 'Only': return 'bg-lime-600/50';
    case 'Missing': return 'bg-lime-600/10';
    default: return '';
  }
}
```

`getLineClass` ordnet jedem Diff-Status eine CSS-Klasse zu und macht Unterschiede visuell erkennbar.

HeaderComponent

Die HeaderComponent zeigt den Kopfbereich der Anwendung und enthält die Navigation zurück zur Startseite.

ButtonComponent

Die ButtonComponent ist eine generische UI-Komponente für wiederverwendbare Buttons mit Click-Event und Disabled-Zustand.

TabButtonComponent

Die TabButtonComponent kapselt die Darstellung und Klickbehandlung eines einzelnen Tabs. Sie wird in der Vergleichsansicht verwendet, um den Tabwechsel einheitlich und wiederverwendbar umzusetzen.

5.1.4 Routes

Das Routing im Frontend definiert die Navigation zwischen den einzelnen Ansichten der Anwendung. Dabei wird festgelegt, welche Komponente (siehe Abschnitt 2.1.3) unter welchem Pfad geladen wird.

Im aktuellen Frontend sind genau drei Routing-Pfade definiert:

- `/` als Standardpfad mit Weiterleitung auf `/home`
- `/home` für die Startseite mit Projektauswahl
- `/compare` für die Vergleichsansicht

AppRoutingModule

Im zentralen App-Routing wird die Standardnavigation definiert. Beim Aufruf der Anwendung ohne Pfadangabe erfolgt eine Weiterleitung auf die Startseite.

```
const ROUTES: Routes = [
  {
    path: '',
    redirectTo: 'home',
    pathMatch: 'full'
  }
];

@NgModule({
  imports: [
    RouterModule.forRoot(ROUTES, { relativeLinkResolution: 'legacy' }),
    HomeRoutingModule,
    CompareRoutingModule
  ],
  exports: [RouterModule]
})
```

Die Konfiguration sorgt dafür, dass die Anwendung immer mit der Home-Ansicht startet. Zusätzlich werden die Routing-Module der einzelnen Bereiche eingebunden.

HomeRoutingModule

Die Route der Home-Ansicht stellt den Einstiegspunkt für die Projektauswahl dar.

```
const ROUTES: Routes = [
  {
    path: 'home',
    component: HomeComponent
  }
];
```

Über diese Route wird die Komponente geladen, in der beide Projektpfade ausgewählt werden.

CompareRoutingModule

Die Compare-Route lädt die zentrale Vergleichsansicht der Anwendung.

```
const ROUTES: Routes = [
  {
    path: 'compare',
    component: CompareComponent
  }
];
```

Diese Route wird nach erfolgreicher Projektauswahl aufgerufen und dient als Container für Overview-, Files- und SAP-Ansicht.

Navigation mit Query-Parametern

Die Route `/compare` benötigt zwei Eingabewerte, damit der Vergleich gestartet werden kann: den Pfad des ersten Projekts und den Pfad des zweiten Projekts. Diese Werte werden beim Navigationsaufruf als Query-Parameter übergeben. Im konkreten Fall sind das die Parameter `project1` und `project2`.

```
public navigateToCompare(): void {
  if (!this.filePathProject1 || !this.filePathProject2) return;

  this.router.navigate(['/compare'], {
    queryParams: {
      project1: this.filePathProject1,
      project2: this.filePathProject2
    }
  });
}
```

```
constructor(private route: ActivatedRoute) {
  this.route.queryParams.subscribe(params => {
    this.project1 = params['project1'];
    this.project2 = params['project2'];
  });
}
```

Damit werden die in der Home-Ansicht gewählten Pfade an die Compare-Komponente übergeben und dort für den Datenabruf verwendet.

5.1.5 Herausforderungen

Bei der Implementierung der Webanwendung traten verschiedene technische Herausforderungen auf, die sowohl die Datenverarbeitung als auch die Benutzeroberfläche betrafen.

Eine wichtige Herausforderung war die Verarbeitung der Daten aus dem Backend. Die Umwandlung der JSON-Daten in die passenden Typen funktionierte nicht immer fehlerfrei. Besonders bei Enums und verschachtelten Objekten kam es zu Problemen. Deshalb mussten zusätzliche Datentypen eingeführt bzw. angepasst werden.

Auch die Zustandsverwaltung im Frontend war anfangs schwierig. Daten wurden teilweise zu spät geladen oder nicht richtig aktualisiert, wodurch die Benutzeroberfläche falsche Zustände angezeigt hat. Durch den Einsatz von `BehaviorSubjects` konnte dieses Problem gelöst werden, damit Änderungen automatisch an die Komponenten (siehe Abschnitt 2.1.3) weitergegeben werden.

Bei der Benutzeroberfläche gab es vor allem bei der Baumansicht Probleme. Angular Material schränkte die Gestaltung und Größenanpassung teilweise ein. Außerdem war es schwierig, zwei Baumansichten gleichzeitig zu synchronisieren, beispielsweise beim Scrollen oder beim Öffnen von Ordnern.

Ein weiterer aufwendiger Teil war der Datei-Vergleich. Unterschiede mussten nicht nur zwischen Zeilen, sondern auch innerhalb einzelner Zeilen erkannt und dargestellt werden. Zusätzlich gab es Darstellungsprobleme, zum Beispiel mit Hintergrundfarben beim Scrollen, die durch Anpassungen mittels TailwindCSS (siehe Abschnitt 2.4.1) behoben wurden.

Die Einbindung von Electron (siehe Abschnitt 2.4.1) brachte ebenfalls einige Schwierigkeiten mit sich. Vor allem der Zugriff auf lokale Dateien und das Öffnen von Ordnern erforderten eine spezielle Umsetzung, zudem wurde ein zusätzlicher Backend-Endpunkt implementiert.

Insgesamt wurde deutlich, dass die Zusammenarbeit zwischen Frontend und Backend sowie die Synchronisation der Oberfläche besonders aufwendig waren. Durch schrittweises Testen konnten die Probleme aber behoben werden.

5.2 Backend

Im Architekturkapitel wurde beschrieben, dass das Backend die Vermittlerrolle zwischen Benutzeroberfläche, Dateisystem und SAP übernimmt (siehe Abschnitt 4.3). Eine Anfrage erreicht zuerst die REST-Schnittstelle, wird anschließend in der Service-Schicht verarbeitet, greift dort auf unterschiedliche Datenquellen zu und liefert am Ende ein klar strukturiertes Ergebnis an das Frontend zurück. Diese Aufteilung folgt den im Theorieteil beschriebenen Prinzipien von REST, DTOs, Interfaces und Services (siehe Abschnitt 2.1.7, Abschnitt 2.1.5, Abschnitt 2.1.8 und Abschnitt 2.1.4).

Die Frontend-Typen wurden im vorderen Teil dieses Implementierungskapitels erläutert (siehe Abschnitt 5.1.1). Da sich diese teilweise stark mit den Typen im Backend überschneiden, werden im Folgenden nur jene Klassen näher behandelt, die für das serverseitige Einlesen, Verarbeiten und Zurückgeben der Daten erforderlich sind.

5.2.1 Datenbeschaffung und Klassen

Die Datenbeschaffung des Backends beginnt nicht erst beim eigentlichen Vergleich, sondern beim Start der Anwendung. In `Program.cs` werden die Grundbausteine der Anwendung registriert: Controller, JSON-Serialisierung, CORS, Swagger sowie die Service-Implementierungen. Gleichzeitig wird NLog initialisiert, damit ab dem Anwendungsstart nachvollziehbar ist, welche Schritte im System ausgeführt wurden.

```
1  Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
2  LogManager.Setup().LoadConfigurationFromFile("NLog.config");
3  LogManager.Configuration.Variables["StartupPath"] = AppDomain.CurrentDomain.BaseDirectory;
4
5  builder.Services.AddControllers()
6      .AddJsonOptions(options =>
7      {
8          options.JsonSerializerOptions.Converters.Add(new JsonStringEnumConverter());
9      });
10
11 builder.Services.AddScoped<ICompareService, CompareService>();
12 builder.Services.AddScoped<IUtilityService, UtilityService>();
13
14 app.UseCors("AllowAll");
15 app.MapControllers();
```

Mehrere Entscheidungen sind an dieser Stelle für den weiteren Ablauf wesentlich. Erstens sorgt der `JsonStringEnumConverter` dafür, dass Enums nicht als numerische Werte, sondern als lesbare Strings serialisiert werden. Das ist insbesondere beim Status von Dateien und Diff-Zeilen relevant, weil das Frontend dadurch direkt mit Bezeichnungen wie `Same`, `Different`, `Only` oder `Missing` arbeiten kann. Zweitens wird über Dependency Injection festgelegt, welche konkrete Klasse hinter den Interfaces `ICompareService` und `IUtilityService` steckt. Dadurch bleiben

Controller und Business-Logik lose gekoppelt, was die Wartbarkeit erhöht und der im Theorieteil beschriebenen Service-Orientierung entspricht.

Controller als Einstieg in den Ablauf

Jede Vergleichsanfrage startet in einem Controller. Für den eigentlichen Projektvergleich ist der `CompareController` zuständig, während der `UtilityController` systemnahe Zusatzfunktionen wie das Öffnen eines Ordners im Windows Explorer bereitstellt. Die Controller bilden damit die äußere API-Schicht der Anwendung.

```
[ApiController]
[Route("api/[controller]")]
public class CompareController(ICompareService compareService) : ControllerBase
{
    private readonly ICompareService _compareService = compareService;

    [HttpGet("Projects/Overview")]
    public IActionResult CompareProjectsOverview([FromQuery] CompareProjectsRequest
        request) { ... }

    [HttpGet("Projects/FileStructure")]
    public IActionResult CompareProjectsFileStructure([FromQuery] CompareProjectsRequest
        request) { ... }

    [HttpGet("Projects/Sap")]
    public IActionResult CompareProjectsSap([FromQuery] CompareProjectsRequest request) {
        ... }

    [HttpGet("Files")]
    public IActionResult CompareFiles([FromQuery] CompareFilesRequest request) { ... }
}
```

Die Controller enthalten bewusst keine umfangreiche Vergleichslogik. Ihre Aufgabe besteht darin, Requests entgegenzunehmen, die Eingaben grundlegend zu prüfen und die Anfrage an die Service-Schicht weiterzugeben. Damit bleiben sie schlank, was sowohl die Lesbarkeit als auch die Testbarkeit verbessert. Gleichzeitig wird im Controller klar sichtbar, welche Endpunkte die Anwendung überhaupt anbietet. Aus Dokumentationssicht ist das hilfreich, weil sich die Backend-Funktionalität an genau diesen Endpunkten orientiert: Overview, Files, SAP und der direkte Datei-Diff.

Für die Vergleichsoperationen wird `[FromQuery]` verwendet, da die Parameter bei `GET`-Requests über die URL übergeben werden. Der `OpenFolder`-Endpunkt nutzt hingegen `[FromBody]`, weil die Daten bei der `POST`-Anfrage im Request-Body gesendet werden.

Request-Modelle für die Eingabe über HTTP

Für die Übergabe der Eingabedaten an das Backend werden eigene Request-Modelle verwendet. Sie beschreiben, welche Informationen ein Endpunkt aus einer HTTP-Anfrage entgegennimmt. Im vorliegenden Fall handelt es sich dabei um Pfadangaben für Projekt- oder Dateivergleiche beziehungsweise um den Pfad eines zu öffnenden Ordners.

```
public class CompareProjectsRequest
{
    public required string ProjectPath1 { get; set; }
    public required string ProjectPath2 { get; set; }
}

public class CompareFilesRequest
{
    public required string FilePath1 { get; set; }
    public required string FilePath2 { get; set; }
}

public class OpenFolderRequest
{
    public required string FolderPath { get; set; }
}
```

Die Request-Modelle fassen die Eingaben einer Anfrage in strukturierter Form zusammen und erleichtern dadurch die Weiterverarbeitung im Controller. Gleichzeitig stellen sie sicher, dass das Frontend die erforderlichen Daten in geeigneter Form übermittelt.

Modelle für Rückgabe und interne Verarbeitung

Neben den Request-Modellen kommen im Backend weitere Modelle zum Einsatz, die für die Verarbeitung der Daten und für die Rückgabe der Ergebnisse benötigt werden. Dazu zählen etwa Strukturen für Verzeichnisbäume, Vergleichsergebnisse oder erkannte Unterschiede in Dateien.

Diese Modelle bilden nicht die Eingabe einer HTTP-Anfrage ab, sondern die Daten, mit denen das Backend intern arbeitet oder die es als Ergebnis an den Client zurückliefert. Dadurch wird die Implementierung klar gegliedert: Während Request-Modelle den Eingang einer Anfrage strukturieren, beschreiben die übrigen Modelle die verarbeiteten Daten und die erzeugten Resultate.

Die eigentlichen Datenquellen des Backends

Inhaltlich arbeitet das Backend mit zwei unterschiedlichen Datenquellen. Die erste Datenquelle ist das Dateisystem. Von dort stammen die Projektordner und Dateien, die für den Diff gelesen werden. In den Projektordnern findet man auch jeweils die `ProjectInfo.json`-Datei. Sie liefert Metadaten, etwa den Projektnamen oder die `FabNumber`, die später für den SAP-Zugriff relevant wird. Weiters gibt es auch noch die `.fcl`-Dateien, in denen Referenzen auf andere Dateien innerhalb des Projekts enthalten sind. Die zweite Datenquelle ist schließlich SAP, das über die Fertigungsnummer zusätzliche ERP-Daten bereitstellt.

Dadurch entsteht ein Backend, das nicht nur Daten durchreicht, sondern mehrere technische Quellen zu einer gemeinsamen Sicht zusammenführt. Jeder Vergleichsmodus der Anwendung greift auf einen anderen Ausschnitt derselben Projektwelt zu. Der Overview-Vergleich nutzt

`ProjectInfo.json`, der Strukturvergleich arbeitet auf Dateisystemebene, der SAP-Vergleich verbindet Projektdaten mit einem externen System und der Datei-Diff liest konkrete Dateien zeilenweise ein.

Hilfsklassen für die Datenbeschaffung

Damit die Service-Schicht nicht mit zu vielen Detailaufgaben überladen wird, wurden mehrere Hilfsklassen eingeführt. Die Klasse `Project` übernimmt die Vereinheitlichung des Projektpfads und das Einlesen der Projektinformationen.

```
1  public static string ResolvePath(string path)
2  {
3      if (Directory.Exists(path))
4      {
5          string harddisk0Path = Path.Combine(path, "harddisk0");
6          if (Directory.Exists(harddisk0Path))
7          {
8              return harddisk0Path;
9          }
10         return Path.GetFullPath(path);
11     }
12     return path;
13 }
14
15 public static ProjectInfo? GetInfo(string projectPath)
16 {
17     string projectInfoPath = Path.Combine(projectPath, "ProjectData", "ProjectInfo.json");
18     if (!File.Exists(projectInfoPath))
19     {
20         return null;
21     }
22
23     ProjectInfo piInstance = new();
24     return piInstance.GetProjectInfo(projectInfoPath);
25 }
```

Die Methode `ResolvePath` löst ein praktisches Problem, das im Projektkontext häufig auftritt: Nicht jeder übergebene Ordner ist direkt der fachliche Projektwurzelordner. Enthält ein Projektverzeichnis noch einen Unterordner `harddisk0`, wird dieser als eigentliche Basis verwendet. Erst danach beginnt der weitere Vergleich. Ohne diese Normalisierung würden viele nachgelagerte Operationen an unterschiedlichen Wurzelebenen arbeiten und dadurch falsche Ergebnisse liefern.

Eine weitere wichtige Hilfsklasse ist `Settings`. Sie lädt beim Start eine XML-Konfiguration, in der eine Blacklist von Dateimustern hinterlegt ist. Diese Blacklist wird anschließend im Helper `FileBlacklist` geladen. Dateien, die nur technische Artefakte darstellen und für den fachlichen Vergleich keinen Mehrwert besitzen, werden dadurch frühzeitig ausgeschlossen. Das macht den Strukturvergleich nicht nur übersichtlicher, sondern reduziert auch unnötige Dateizugriffe.

```
<Data>
  <Blacklist>
    <Item>.*\.FUD$</Item>
    <Item>.*\.FUL$</Item>
    <Item>.*\.dbg$</Item>
    <Item>.*\.md5$</Item>
    <Item>.*\.SUL$</Item>
  </Blacklist>
</Data>
```

Zusätzlich übernimmt die Klasse `Fcl` das Parsen von `.fcl`-Dateien. Diese Dateien verweisen innerhalb eines Projekts auf weitere Dateien, die fachlich relevant sein können. Für die spätere Baumdarstellung ist diese Information nützlich, weil damit sichtbar gemacht werden kann, ob eine Datei in einer FCL referenziert wird oder nicht.

```
public static void AddReferencedFiles(string fclPath, string projectPath, HashSet<string>
    targetSet)
{
    LineListProvider lp = new(fclPath);
    lp.ReadCleaned();

    foreach (string line in lp.LineList)
    {
        if (string.IsNullOrWhiteSpace(line))
            continue;

        if (line.StartsWith("#include", StringComparison.OrdinalIgnoreCase))
            continue;

        string varfil = StringOperations.GetToken(line, 1, " ", true);
        varfil = varfil.Replace("/", "\\");
        ...
        string relative = Path.GetRelativePath(projectPath, normalized);
        targetSet.Add(relative);
    }
}
```

Die Hilfsklassen machen damit deutlich, dass Datenbeschaffung im Backend nicht auf einen einzelnen Zugriff reduziert werden kann. Vielmehr wird die fachlich relevante Ausgangsbasis in mehreren kleinen, spezialisierten Schritten hergestellt: Pfad normalisieren, Projektinformationen laden, Blacklist einlesen, FCL-Referenzen extrahieren und gegebenenfalls SAP-Daten abrufen. Erst auf dieser Grundlage kann die eigentliche Business Logic sinnvoll arbeiten.

5.2.2 Business Logic

Nachdem im vorherigen Abschnitt die Datenquellen und Klassen eingeordnet wurden, folgt nun die eigentliche fachliche Verarbeitung. Das Zentrum dieser Verarbeitung ist der `CompareService`. Er implementiert das Interface `ICompareService` und stellt damit die vier relevanten Vergleichsoperationen für die Benutzeroberfläche zur Verfügung.

```
public class CompareService : ICompareService
{
    public List<TableDifference> CompareProjectInfos(string path1, string path2) { ... }
    public FileStructureComparison CompareFileStructure(string path1, string path2) { ... }
    public List<TableDifference> CompareSapData(string path1, string path2) { ... }
    public FileComparison CompareFiles(CompareFilesRequest request) { ... }
}
```

Der `CompareService` ist damit keine Klasse, die alle Detailarbeit selbst erledigt. Seine Arbeit liegt darin, den Ablauf zu steuern: Zuerst werden Eingaben vorbereitet, danach werden spezialisierte Helper aufgerufen, und zuletzt werden die Ergebnisse in die passenden Response-Modelle überführt. Dadurch bleibt die Logik lesbar. Für jeden Vergleichstyp lässt sich die gleiche Grundstruktur erkennen: Eingabe aufbereiten, Datenquellen lesen, Unterschiede berechnen, Ergebnis zurückgeben.

Overview-Vergleich als Vergleich von ProjectInfo-Daten

Der einfachste Vergleichstyp ist der Overview-Vergleich. Hier werden nicht komplette Dateien oder Ordnerbäume verglichen, sondern nur die in `ProjectInfo.json` gespeicherten Projektdaten. Dieser Vergleichstyp liefert dem Benutzer einen schnellen Überblick über die wichtigsten Unterschiede, ohne dass er zuerst in einzelne Ordner oder Dateien einsteigen muss.

```
1  public List<TableDifference> CompareProjectInfos(string path1, string path2)
2  {
3      path1 = Project.ResolvePath(path1);
4      path2 = Project.ResolvePath(path2);
5
6      ProjectInfo? projectInfos1 = Project.GetInfo(path1);
7      List<KeyValueEntry> flat1 = projectInfos1 != null ?
           JsonKeyValue.ProjectInfoToKeyValue(projectInfos1) : [];
8
9      ProjectInfo? projectInfos2 = Project.GetInfo(path2);
10     List<KeyValueEntry> flat2 = projectInfos2 != null ?
           JsonKeyValue.ProjectInfoToKeyValue(projectInfos2) : [];
11
12     return JsonKeyValue.CompareKeyValueLists(flat1, flat2);
13 }
```

Zuerst werden beide Projektpfade normalisiert. Danach werden die jeweiligen `ProjectInfo`-Objekte geladen. Anschließend wird die strukturierte JSON-Repräsentation in eine flache Liste aus Key-Value-Paaren umgewandelt. Dadurch wird der Vergleich übersichtlicher, da sich die Webanwendung nicht mit verschachtelten Listen beschäftigen muss.

Diese Umwandlung übernimmt die Klasse `JsonKeyValue`. Sie traversiert rekursiv durch das JSON-Objekt und erzeugt daraus Einträge mit Pfaden in Punktnotation. Zusätzlich werden Arrays vereinheitlicht und bestimmte technische Eigenschaften aus dem Ergebnis entfernt.

```
public static List<KeyValueEntry> ProjectInfoToKeyValue(ProjectInfo project)
{
    List<KeyValueEntry> result = ObjectToKeyValue(project);

    for (int i = result.Count - 1; i >= 0; i--)
    {
        KeyValueEntry kv = result[i];

        if (kv.Key.EndsWith(".Changeable") || kv.Key.EndsWith(".Visible"))
        {
            result.RemoveAt(i);
            continue;
        }

        if (kv.Key.EndsWith(".Value"))
        {
            kv.Key = kv.Key.Replace(".Value", string.Empty);
        }
    }

    return result;
}
```

Eigenschaften wie `Changeable` oder `Visible` sind für den Vergleich zweier Projekte im Overview-Tab meist nicht aussagekräftig, weil sie primär Darstellungs- oder Metadatencharakter besitzen. Durch das Entfernen solcher Anteile bleibt der Vergleich auf jene Inhalte konzentriert, die der Benutzer tatsächlich als Projektunterschied wahrnimmt. Ebenso wird das Suffix `.Value` entfernt, um den Vergleich übersichtlicher zu gestalten.

Nachdem beide Listen von `KeyValueEntry`-Objekten vorliegen, werden sie zusammengeführt. Für jeden Key entsteht anschließend ein `TableDifference`-Objekt, das beide Werte und ein `IsDifferent`-Flag enthält. Dieser Ansatz hat den Vorteil, dass fehlende Werte automatisch als leere Einträge sichtbar werden. Der Benutzer erkennt also nicht nur geänderte Werte, sondern auch Informationen, die auf einer Seite vollständig fehlen.

Abbildung 24 visualisiert diesen Ablauf noch einmal als Flowchart.

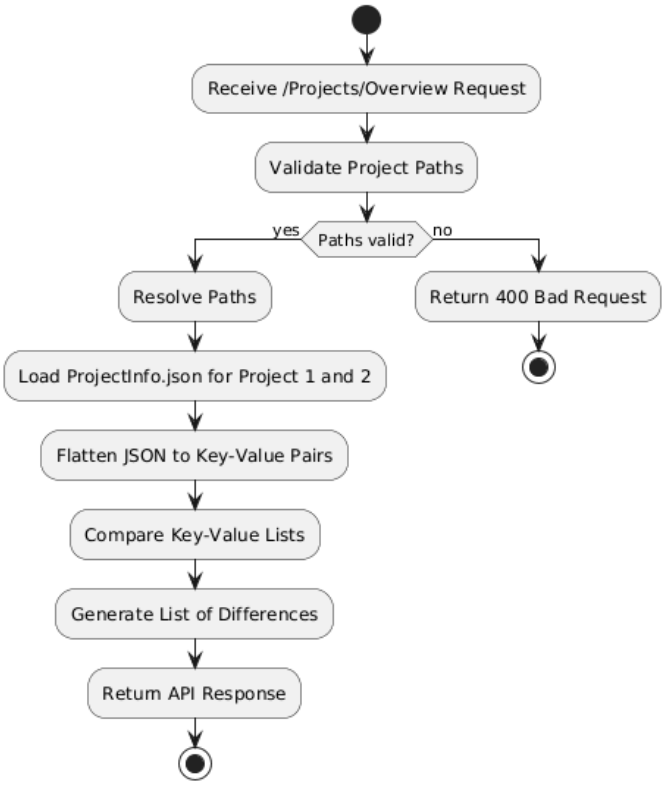


Abbildung 24: Overview Comparison Flowchart

Vergleich der Dateistruktur

Der Vergleich der Dateistruktur ist deutlich umfangreicher als der Overview-Vergleich. Hier reicht es nicht, einzelne Metadaten zu lesen. Stattdessen muss das Backend komplette Projektordner rekursiv einlesen, irrelevante Dateien ausfiltern, Referenzen aus `.fcl`-Dateien sammeln, Inhalte über Hashwerte vergleichbar machen und die beiden resultierenden Bäume schließlich so aufeinander abbilden, dass das Frontend eine synchrone Gegenüberstellung darstellen kann.

```
public FileStructureComparison CompareFileStructure(string path1, string path2)
{
    if (string.IsNullOrWhiteSpace(path1) || string.IsNullOrWhiteSpace(path2)
        || !Directory.Exists(path1) || !Directory.Exists(path2))
    {
        return new FileStructureComparison
        {
            Project1AbsolutePath = path1,
            Project1Files = [],
            Project2AbsolutePath = path2,
            Project2Files = [],
        };
    }

    string resolvedPath1 = Project.ResolvePath(path1);
    string resolvedPath2 = Project.ResolvePath(path2);

    HashSet<string> fclPaths1 = new(StringComparer.OrdinalIgnoreCase);
    HashSet<string> fclPaths2 = new(StringComparer.OrdinalIgnoreCase);

    FileNode tree1 = FileTree.Build(resolvedPath1, resolvedPath1, fclPaths1);
    FileNode tree2 = FileTree.Build(resolvedPath2, resolvedPath2, fclPaths2);

    FileTree.MarkFilesUsedInFcl(tree1, fclPaths1);
    FileTree.MarkFilesUsedInFcl(tree2, fclPaths2);

    (FileNode, FileNode) comparedTrees = FileTree.Compare(tree1, tree2);

    FileNodeResponse comparedTree1 = FileTree.MapToResponseFileNode(comparedTrees.Item1);
    FileNodeResponse comparedTree2 = FileTree.MapToResponseFileNode(comparedTrees.Item2);

    return new FileStructureComparison { ... };
}
```

An dieser Methode lässt sich erkennen, dass die eigentliche Schwierigkeit nicht im Rückgabeobjekt, sondern in den vorbereitenden Zwischenschritten liegt. Die Service-Methode stößt dabei die einzelnen Schritte nacheinander an, während der fachliche Kern in `FileTree` liegt.

Aufbau des Dateibaums Beim Aufbauen des Dateibaums wird jede Datei und jedes Verzeichnis rekursiv erfasst. Die Implementierung arbeitet dabei teilweise parallel, um insbesondere bei größeren Projekten das Traversieren des Dateisystems zu beschleunigen.

```
public static FileNode Build(string path, string rootPath, HashSet<string> fclPaths)
{
    FileNode node = new()
    {
        Name = Path.GetFileName(path),
        IsDirectory = true,
        RelativePath = Path.GetRelativePath(rootPath, path),
        Children = []
    };

    ConcurrentBag<FileNode> childNodes = [];

    Parallel.ForEach(Directory.GetDirectories(path), dir =>
    {
        FileNode subNode = Build(dir, rootPath, fclPaths);
        childNodes.Add(subNode);
    });

    Parallel.ForEach(Directory.GetFiles(path), file =>
    {
        string fileName = Path.GetFileName(file);
        if (FileBlacklist.IsMatch(fileName))
            return;

        if (Path.GetExtension(fileName) == ".fcl")
        {
            lock (fclPaths)
            {
                Fcl.AddReferencedFiles(file, rootPath, fclPaths);
            }
        }

        FileNode fileNode = new()
        {
            Name = fileName,
            IsDirectory = false,
            RelativePath = Path.GetRelativePath(rootPath, file),
            Hash = ComputeFileHash(file)
        };

        childNodes.Add(fileNode);
    });

    node.Children = SortNodes(childNodes);
    return node;
}
```

Die Methode erzeugt für jedes Verzeichnis einen `FileNode`. Für Dateien wird zusätzlich ein SHA-256-Hash berechnet. Dieser Hash ist entscheidend, weil sich damit Inhalte effizient vergleichen lassen, ohne Dateien während der eigentlichen Vergleichsphase mehrfach vollständig lesen zu müssen. Gleichzeitig werden Dateien, die der Blacklist entsprechen, direkt verworfen. Diese Filterung reduziert Rauschen im Ergebnis und verhindert, dass rein technische Artefakte den Vergleich verfälschen.

Ein weiterer interessanter Punkt ist die Verarbeitung von `.fcl`-Dateien. Sie werden beim Traversieren erkannt und sofort analysiert. Die daraus extrahierten relativen Dateipfade werden in einer `HashSet`-Struktur gesammelt. Später kann damit bei jedem Knoten im Baum markiert werden, ob er in einer FCL referenziert wird. Fachlich ist das nützlich, weil der Benutzer auf einen Blick erkennt, welche Dateien in einer solchen Referenzliste verwendet werden.

Vergleich der beiden Bäume Nachdem beide Projektbäume aufgebaut wurden, müssen sie miteinander in Beziehung gesetzt werden. Der Vergleich erfolgt nicht bloß über die Reihenfolge der Dateien, sondern über Namen und Struktur.

```
1  public static (FileNode node1, FileNode node2) Compare(FileNode n1, FileNode n2)
2  {
3      Dictionary<string, FileNode> dict1 = n1.Children?.ToDictionary(c => c.Name) ?? [];
4      Dictionary<string, FileNode> dict2 = n2.Children?.ToDictionary(c => c.Name) ?? [];
5
6      List<string> allNames = GetSortedCombinedNames(dict1, dict2);
7
8      List<FileNode> list1 = [];
9      List<FileNode> list2 = [];
10
11     AlignAndCompareChildren(allNames, dict1, dict2, list1, list2);
12
13     FileNodeStatus status1 = DetermineDirectoryStatus(list1,
           !string.IsNullOrEmpty(n1.Name));
14     FileNodeStatus status2 = DetermineDirectoryStatus(list2,
           !string.IsNullOrEmpty(n1.Name));
15
16     return (
17         new FileNode { Name = n1.Name, IsDirectory = true, InFcl = n1.InFcl, RelativePath
               = n1.RelativePath, Children = list1, Status = status1 },
18         new FileNode { Name = n2.Name, IsDirectory = true, InFcl = n2.InFcl, RelativePath
               = n2.RelativePath, Children = list2, Status = status2 }
19     );
20 }
```

Die beiden Kinderlisten werden zunächst in Dictionaries überführt. Danach wird eine gemeinsame, sortierte Namensmenge gebildet. Das ist wichtig, weil die UI eine synchrone Gegenüberstellung erwartet: Links und rechts soll jeweils der Eintrag an derselben Position stehen, auch wenn eine Datei nur in einem Projekt vorhanden ist. Um das zu erreichen, erzeugt der Algorithmus bei Bedarf Platzhalterknoten mit dem Status `Missing`. Existiert eine Datei nur auf einer Seite, wird sie auf der anderen Seite als fehlend gespiegelt. Dadurch bleibt die Struktur beider Listen parallel.

Für Dateien wird der Status über den Hashwert bestimmt. Identische Hashwerte führen zu `Same`, unterschiedliche zu `Different`. Für Verzeichnisse wird der Status aus den Statuswerten ihrer Kinder abgeleitet. Zuerst werden also einzelne Dateien verglichen, daraus ergibt sich der Status der übergeordneten Ordner.

Markierung von FCL-Referenzen und Mapping in API-Objekte Nach dem Aufbau und vor dem finalen Mapping werden die gesammelten FCL-Pfade auf den Baum angewendet.

```
public static FileNode MarkFilesUsedInFcl(FileNode node, HashSet<string> fclPaths)
{
    if (node.IsDirectory)
    {
        if (node.Children != null)
        {
            foreach (FileNode child in node.Children)
            {
                MarkFilesUsedInFcl(child, fclPaths);
            }
        }
        return node;
    }

    node.InFcl = fclPaths.Contains(node.RelativePath);
    return node;
}

public static FileNodeResponse MapToResponseFileNode(FileNode node)
{
    return new FileNodeResponse
    {
        Name = node.Name,
        IsDirectory = node.IsDirectory,
        InFcl = node.InFcl ?? false,
        RelativePath = node.RelativePath,
        Status = node.Status ?? FileNodeStatus.Different,
        Children = node.Children?.Select(MapToResponseFileNode).ToList()
    };
}
```

Der erste Schritt versieht jede Datei im Baum mit einer fachlich sichtbaren Information. Der zweite Schritt entfernt dann alle rein internen Details, insbesondere den Hash, aus der API-Antwort. An dieser Stelle wird die zuvor beschriebene Trennung zwischen internen und externen Modellen sichtbar.

Abbildung 25 fasst diesen Ablauf als Flowchart zusammen.

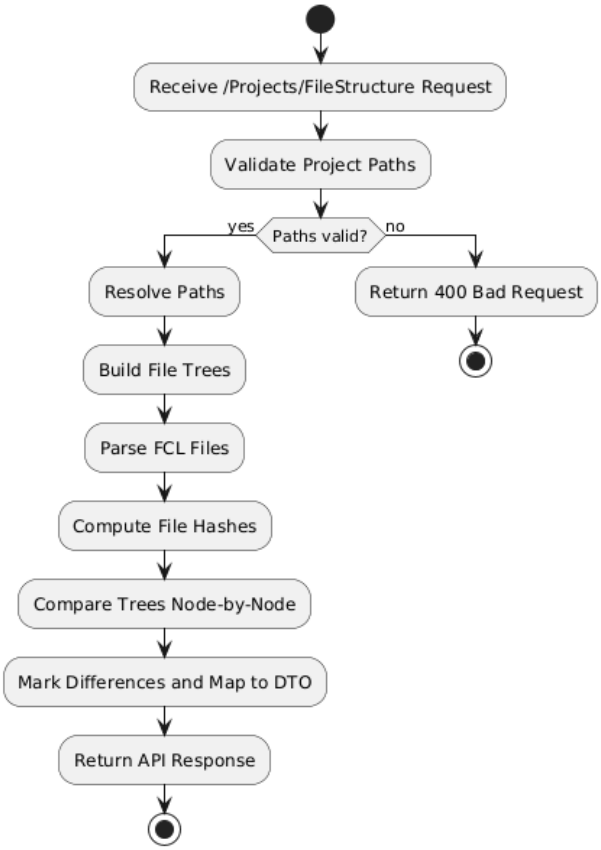


Abbildung 25: File Structure Comparison Flowchart

SAP-Vergleich

Der SAP-Vergleich folgt erneut derselben Grundidee wie die anderen Vergleiche, nutzt aber eine zusätzliche Datenquelle außerhalb des Dateisystems. Die Verbindung zwischen Projekt und SAP wird über die `FabNumber` hergestellt, die aus `ProjectInfo.json` gelesen wird.

```
1  public List<TableDifference> CompareSapData(string path1, string path2)
2  {
3      path1 = Project.ResolvePath(path1);
4      path2 = Project.ResolvePath(path2);
5
6      string? fabNo1 = Project.GetInfo(path1)?.FabNumber;
7      string? fabNo2 = Project.GetInfo(path2)?.FabNumber;
8
9      List<KeyValueEntry> sapData1 = !string.IsNullOrEmpty(fabNo1) ? Sap.GetSapData(fabNo1)
           : [];
10     List<KeyValueEntry> sapData2 = !string.IsNullOrEmpty(fabNo2) ? Sap.GetSapData(fabNo2)
           : [];
11
12     return JsonKeyValue.CompareKeyValueLists(sapData1, sapData2);
13 }
```

Auffällig ist hier, dass der eigentliche Vergleich erneut über `JsonKeyValue.CompareKeyValueLists` läuft. Obwohl die Daten aus unterschiedlichen Quellen stammen, werden sie vor dem Vergleich in ein gemeinsames, einfaches Format überführt. Für den eigentlichen Vergleich spielt es dann keine Rolle mehr, ob die Eingaben ursprünglich aus JSON oder aus SAP kamen.

Die Klasse `Sap` kapselt den Datenabruf aus dem ERP-System. Sie vereinigt dabei Informationen aus zwei Quellen, die Equipment-Daten und die Operations-Daten. Falls derselbe Schlüssel in beiden Quellen vorkommt, wird der Wert aus `EquiData` bevorzugt, sofern er nicht leer ist.

```
1  public static List<KeyValueEntry> GetSapData(string fabNo)
2  {
3      SapData sapData = new(fabNo);
4
5      Dictionary<string, string> equiDict = sapData.OpsEquiData.EquiData.ItemList;
6      Dictionary<string, string> opsDict = sapData.OpsEquiData.OpsData.ItemList;
7
8      HashSet<string> allKeys = [.. equiDict.Keys, .. opsDict.Keys];
9      List<KeyValueEntry> resultList = [];
10
11     foreach (string key in allKeys)
12     {
13         equiDict.TryGetValue(key, out var valueEqui);
14         opsDict.TryGetValue(key, out var valueOps);
15         string finalValue = !string.IsNullOrEmpty(valueEqui) ? valueEqui : (valueOps ??
               string.Empty);
16
17         resultList.Add(new KeyValueEntry { Key = key, Value = finalValue });
18     }
19
20     return resultList;
21 }
```

Im SAP-Tab muss der Benutzer also nicht direkt mit den Rohdaten aus dem ERP-System arbeiten. Stattdessen erhält er eine zusammengeführte Gegenüberstellung, die wie der Overview-Vergleich aufgebaut ist. So bleibt die Oberfläche konsistent, auch wenn im Hintergrund eine zusätzliche externe Quelle genutzt wird.

Die Visualisierung in Abbildung 26 zeigt den Ablauf: Projektinformationen werden gelesen, Fertigungsnummern extrahiert, SAP-Daten geladen, die Daten vereinheitlicht, anschließend verglichen und schließlich zurückgegeben.

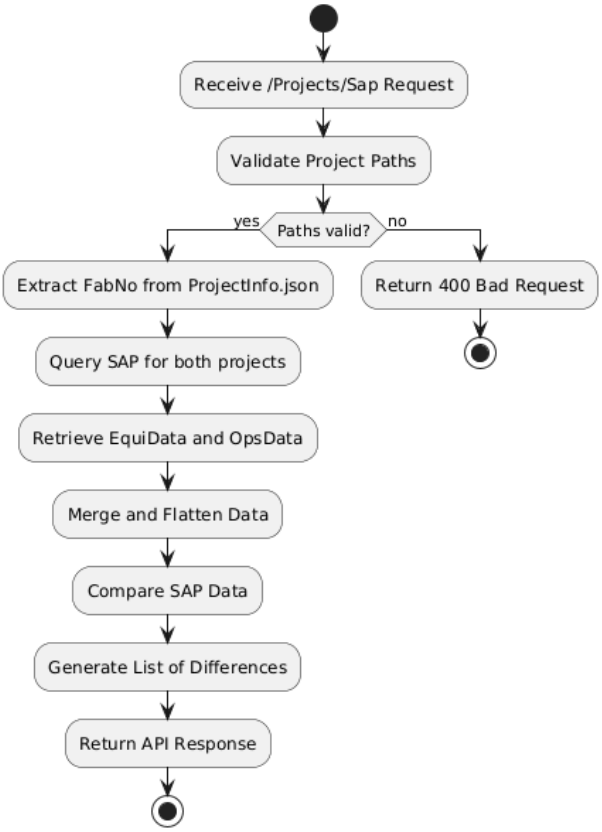


Abbildung 26: SAP Comparison Flowchart

Datei-Inhaltsvergleich

Der vierte Vergleichstyp betrifft nicht mehr ganze Projekte, sondern zwei konkrete Dateien. Während der Strukturvergleich nur feststellt, ob Dateien gleich oder verschieden sind, liefert der Datei-Inhaltsvergleich eine detailgenaue Erklärung dieser Unterschiede. Hierzu genügt ein einfacher Hashvergleich nicht mehr; stattdessen müssen die Inhalte zeilenweise und bei Bedarf sogar innerhalb einzelner Zeilen verglichen werden.

```
public FileComparison CompareFiles(CompareFilesRequest request)
{
    List<string> lines1 = [];
    List<string> lines2 = [];

    if (File.Exists(request.FilePath1))
    {
        LineListProvider lineListProvider1 = new(request.FilePath1);
        lines1 = lineListProvider1.Read();
    }

    if (File.Exists(request.FilePath2))
    {
        LineListProvider lineListProvider2 = new(request.FilePath2);
        lines2 = lineListProvider2.Read();
    }

    (List<DiffLine>, List<DiffLine>) result = FileCompare.Compare([.. lines1], [..
        lines2]);

    return new FileComparison
    {
        File1Lines = result.Item1,
        File2Lines = result.Item2
    };
}
```

Der Service liest also beide Dateien zunächst in Zeilenlisten ein und delegiert danach an den Helper `FileCompare`. Die Methode bleibt auch funktional, wenn eine der beiden Dateien fehlt. In diesem Fall bleibt die entsprechende Zeilenliste leer. Das Diff kann trotzdem berechnet werden, wodurch das Frontend eine saubere Gegenüberstellung zwischen vorhandenen und fehlenden Inhalten darstellen kann.

Die eigentliche Logik des Diffs ist im Helper `FileCompare` gekapselt. Dort wird zunächst ein zeilenweiser Vergleich mit Hilfe einer Diff-Library durchgeführt, die auf dem Myers-Differenzalgorithmus basiert (siehe Abschnitt 2.4.2). Aus Sicht der Anwendung ist weniger die Library selbst entscheidend, sondern wie ihr Ergebnis in ein nutzbares Format überführt wird.

```
public static (List<DiffLine> list1, List<DiffLine> list2) Compare(string[] lines1,
    string[] lines2)
{
    List<DiffLine> list1 = [];
    List<DiffLine> list2 = [];

    Diff.Item[] diffResults = Diff.DiffText(string.Join("\n", lines1), string.Join("\n",
        lines2), true, true, false);
    ...
    ApplyDiffBlock(diffResults[i], lines1, lines2, list1, list2);
    ...
    return (list1, list2);
}
```

Die Methode erzeugt zwei synchrone Listen von `DiffLine`-Objekten. Unveränderte Zeilen werden mit dem Status `Same` versehen, nur auf einer Seite vorhandene Zeilen mit `Only` beziehungsweise `Missing`. Wenn zwei Zeilen einander zugeordnet werden können, aber inhaltlich unterschiedlich sind, folgt ein zweiter, feinerer Vergleichsschritt auf Tokenebene.

```
private static void AddDiffListItem(List<DiffLine> listA, List<DiffLine> listB,
    int lineNumberA, int lineNumberB, string contentA, string contentB)
{
    string[] tokens1 = Tokenize(contentA);
    string[] tokens2 = Tokenize(contentB);

    Diff.Item[] diffResults = Diff.DiffText(string.Join("\n", tokens1), string.Join("\n",
        tokens2), false, false, false);

    DiffContent[] contentList1 = StringToDiffContentArr(tokens1);
    DiffContent[] contentList2 = StringToDiffContentArr(tokens2);
    ...
}
```

Diese zusätzliche Tokenisierung ist für die Benutzerfreundlichkeit der Anwendung sehr wichtig. Ein reiner Zeilenvergleich würde nur anzeigen, dass eine Zeile geändert wurde, nicht aber, an welcher Stelle innerhalb der Zeile die Änderung liegt. Durch den zweiten Vergleichsschritt können auch Teilbereiche einer Zeile markiert werden.

Es wird ein selbst entwickelter Regex-Tokenizer verwendet, der speziell für Code und codeähnliche Inhalte optimiert ist und die Zeilen in sinnvolle Einheiten wie Wörter, Pfadangaben oder einzelne Sonderzeichen zerlegt. Dadurch wird verhindert, dass der Vergleich nur auf Zeichenebene stattfindet, und stattdessen an logisch zusammengehörigen Bestandteilen orientiert ist. Das ist insbesondere bei Code oder Konfigurationswerten hilfreich, da Änderungen so gezielter erkannt und dargestellt werden können.

Nach dem Markieren der geänderten Tokens werden benachbarte Segmente mit gleichem Markierungsstatus wieder zusammengeführt. Dadurch wird die Datenstruktur kompakter und die übertragene Datenmenge reduziert. Würde jedes einzelne Token separat an das Frontend übergeben, entstünde eine unnötig feine Aufteilung.

```
private static List<DiffContent> CompressDiffContent(DiffContent[] tempArray)
{
    List<DiffContent> result = [];
    foreach (DiffContent item in tempArray)
    {
        if (result.Count > 0 && result[^1].IsMarked == item.IsMarked)
        {
            result[^1].Content += item.Content;
        }
        else
        {
            result.Add(new DiffContent { Content = item.Content.ToString(), IsMarked =
                item.IsMarked });
        }
    }
    return result;
}
```

Insgesamt ergibt sich daraus ein mehrstufiger Diff-Prozess: Dateien einlesen, zeilenweise vergleichen, geänderte Zeilen tokenisieren, Markierungen setzen und am Ende zwei parallele `DiffLine`-Listen zurückgeben. In Abbildung 27 ist dieser Ablauf als Flowchart dargestellt.

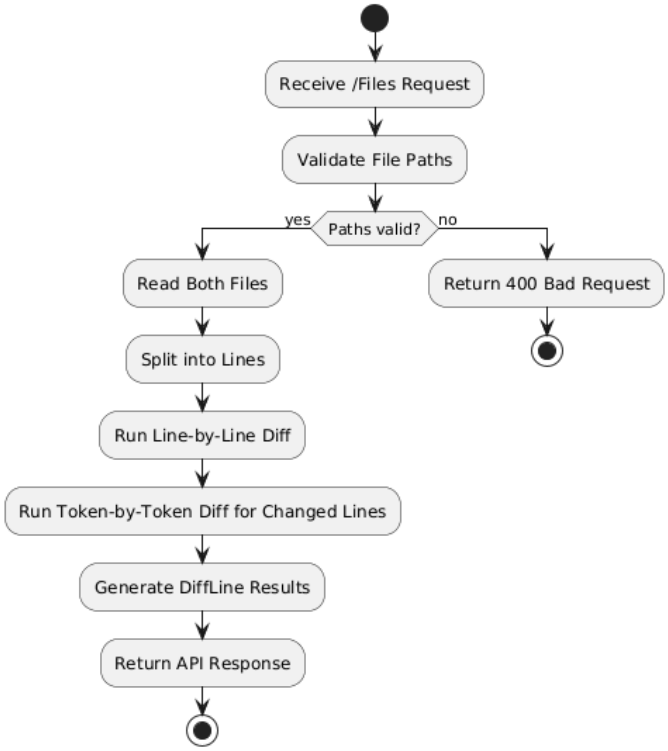


Abbildung 27: File Comparison Flowchart

Systemnahe Hilfsfunktion

Neben den eigentlichen Vergleichsoperationen gibt es noch einen kleinen, aber praxisrelevanten Zusatzendpoint: Das Öffnen eines Ordners im Explorer. Auch diese Funktion wird bewusst über Service und Controller geführt und nicht direkt im Frontend umgesetzt. Dadurch bleibt die Verantwortung für betriebssystemspezifische Aktionen im Backend.

```
1  public bool OpenFolder(string path)
2  {
3      if (!Directory.Exists(path))
4          return false;
5
6      path = Path.GetFullPath(path);
7
8      ProcessStartInfo processInfo = new()
9      {
10         FileName = "explorer.exe",
11         Arguments = $"\"{path}\"",
12         UseShellExecute = true
13     };
14
15     Process? process = Process.Start(processInfo);
16     return process != null;
17 }
```

Die Methode prüft den Pfad, normalisiert ihn und startet anschließend `explorer.exe`. Der Ablauf ist zwar einfach, zeigt aber gut, warum die Trennung zwischen Frontend und Backend auch bei Hilfsfunktionen sinnvoll ist: Die Webanwendung muss keine Windows-spezifischen Details kennen, sondern löst die Aktion lediglich über die API aus.

Abbildung 28 zeigt auch diesen Ablauf noch einmal in vereinfachter Form.

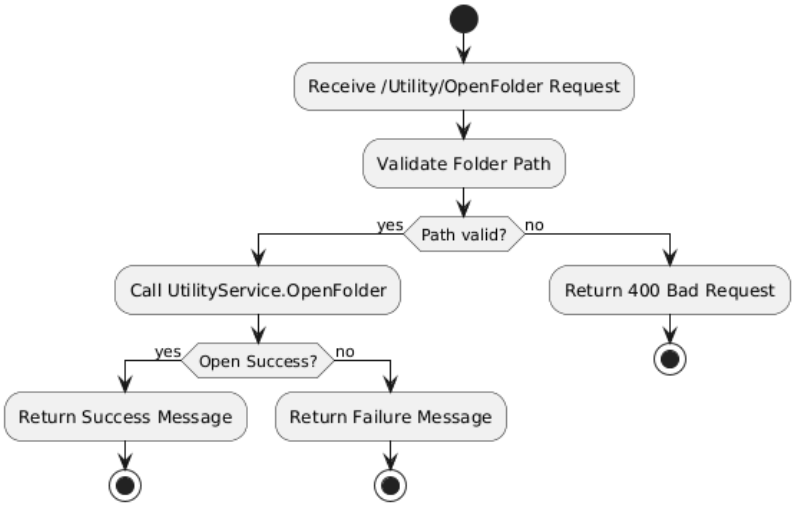


Abbildung 28: Open Folder Flowchart

Zusammenfassung der Business Logic

Über alle vier Vergleichstypen hinweg zeigt sich ein konsistentes Muster. Die Controller definieren die Einstiegspunkte. Der `CompareService` verwaltet den jeweiligen Ablauf. Spezialisierte Helper übernehmen klar abgegrenzte Teilaufgaben wie Pfadauflösung, JSON-Transformation, Dateibaumaufbau, SAP-Datenkonsolidierung oder Diff-Berechnung. Dadurch bleibt die Logik nicht nur modular, sondern auch gut erklärbar: Jede Anfrage durchläuft dieselben Ebenen, auch wenn die eigentliche Verarbeitung je nach Vergleichsmodus unterschiedlich tief ausfällt.

5.2.3 Logging / Fehlerbehandlung

In einer Anwendung, die mit lokalen Dateipfaden, großen Verzeichnisstrukturen und einem externen ERP-System arbeitet, ist Logging nicht nur ein nettes Extra. Es ist ein zentraler Bestandteil, um Abläufe nachvollziehbar zu machen und Fehlerursachen gezielt zu finden. Zwar wurden ergänzend auch Tests mit xUnit implementiert (siehe Abschnitt 2.4.2), diese können jedoch keine vollständige Sicherheit garantieren. Gleichzeitig muss die Fehlerbehandlung so gestaltet sein, dass ungültige Eingaben oder unvollständige Daten nicht zum Absturz der Anwendung führen. Das Backend verfolgt daher folgende Strategie: Eingaben werden möglichst früh validiert, technische Probleme werden geloggt und viele Fehlerfälle führen zu kontrollierten, leeren oder vereinfachten Ergebnissen statt zu einer kompletten Unterbrechung des Vergleichs.

Initialisierung und zentrale Kapselung des Loggings

In `Program.cs` wird NLog initialisiert. Dadurch steht das Logging ab dem Start der Anwendung zur Verfügung und Fehler können von Anfang an erfasst werden.

```
LogManager.Setup().LoadConfigurationFromFile("NLog.config");
LogManager.Configuration.Variables["StartupPath"] = AppDomain.CurrentDomain.BaseDirectory;
NLogging.Instance.Info("ProjectCompare started");
```

Die eigentlichen Log-Aufrufe erfolgen jedoch nicht direkt über NLog, sondern über die Wrapper-Klasse `NLogging`. Diese Klasse kapselt die am häufigsten verwendeten Log-Level zentral.

```
public sealed class NLogging
{
    private readonly Logger _logger;
    public static NLogging Instance => _instance.Value;

    public void Info(string message)  { _logger.Info(message); }
    public void Error(string message) { _logger.Error(message); }
    public void Debug(string message) { _logger.Debug(message); }
    public void Warn(string message)  { _logger.Warn(message); }
    public void Fatal(string message) { _logger.Fatal(message); }
}
```

Diese Kapselung hat zwei Vorteile. Erstens bleibt die Verwendung im gesamten Backend einheitlich. Zweitens könnte die Logging-Technik später ausgetauscht oder erweitert werden, ohne jeden Aufruf im Projekt anpassen zu müssen.

Wie und wohin geloggt wird, ist in `NLog.config` festgelegt.

```
<target name="projectcomparelog" xsi:type="File"
        layout="[${date:format=yyyy-MM\-dd HH\:mm\:ss}] ${message}"
        fileName="${var:StartupPath}\Log\ProjectCompare.log"
        archiveFileName="${var:StartupPath}\Log\ArchiveProjectCompare{####}.log"
        archiveAboveSize="1000000"
        archiveNumbering="Rolling"
        maxArchiveFiles="5"
        encoding="iso-8859-2" />

<logger name="*" minlevel="Debug" writeTo="projectcomparelog" />
```

Das Logging wird dateibasiert durchgeführt. Erreicht die Logdatei eine bestimmte Größe, werden Archivdateien erzeugt. Dieses Rolling-Verfahren ist für eine Anwendung sinnvoll, die in einem Unternehmenskontext mehrfach eingesetzt und getestet wird, weil die Logdaten nachvollziehbar bleiben, ohne unkontrolliert anzuwachsen.

Validierung und Fehlerbehandlung auf Controller-Ebene

Die erste Verteidigungslinie gegen Fehler befindet sich im Controller. Hier werden die eingehenden Pfade auf grundlegende Plausibilität geprüft, bevor überhaupt eine fachliche Verarbeitung startet. Für die projektbezogenen Endpunkte existiert dazu eine gemeinsame Hilfsmethode.

```
1  private BadRequestObjectResult? ValidateProjectPaths(string path1, string path2)
2  {
3      if (string.IsNullOrWhiteSpace(path1) || string.IsNullOrWhiteSpace(path2))
4      {
5          NLogging.Instance.Error("Both project file paths are required.");
6          return BadRequest("Both project file paths are required.");
7      }
8
9      if (!Directory.Exists(path1) || !Directory.Exists(path2))
10     {
11         NLogging.Instance.Error("One or both project paths do not exist or are not
               directories.");
12         return BadRequest("One or both project paths do not exist or are not
               directories.");
13     }
14
15     return null;
16 }
```

Dieser Ansatz reduziert doppelten Code und schafft gleichzeitig eine einheitliche Fehlerkommunikation für mehrere Endpunkte. Der Benutzer beziehungsweise das Frontend erhält früh eine klare Rückmeldung, wenn die übergebenen Projektpfade nicht verwendbar sind. Ähnlich funktioniert der `UtilityController`, der zuerst prüft, ob ein Ordnerpfad vorhanden und gültig ist, bevor ein Explorer-Prozess gestartet wird.

Es wird bewusst auf einfache und direkte Rückmeldungen gesetzt. Statt komplexer Fehlerobjekte werden verständliche Textmeldungen per `BadRequest` zurückgegeben. Für den internen Vergleich von lokalen Projekten ist diese Form ausreichend und gut nachvollziehbar.

Defensives Verhalten in Service- und Helper-Schicht

Nicht jeder Fehlerfall lässt sich im Controller abfangen. Manche Probleme treten erst später auf, etwa wenn eine `ProjectInfo.json` fehlt, eine SAP-Abfrage fehlschlägt oder eine Konfigurationsdatei nicht gelesen werden kann. In diesen Fällen verfolgt das Backend ein defensives Verhalten: Die Probleme werden geloggt, gleichzeitig wird aber versucht, die Anfrage nach Möglichkeit trotzdem in einer sinnvollen reduzierten Form zu beantworten.

Ein Beispiel dafür ist `Project.GetInfo`. Fehlt die Datei `ProjectInfo.json`, liefert die Methode `null` zurück und schreibt eine Fehlermeldung ins Log. Der Service erzeugt in diesem Fall statt eines Absturzes einfach eine leere Liste von `KeyValueEntry`-Objekten. Der spätere Vergleich funktioniert weiterhin und macht durch leere Werte sichtbar, dass auf einer Seite keine Informationen vorhanden waren.

Auch im SAP-Helper ist dieses Muster erkennbar. Der Zugriff auf SAP kann fehlschlagen; deshalb wird der Konstruktoraufruf von `SapData` in einen `try-catch`-Block eingebettet.

```
public static List<KeyValueEntry> GetSapData(string fabNo)
{
    SapData sapData;

    try
    {
        sapData = new(fabNo);
    }
    catch (Exception)
    {
        NLogging.Instance.Error($"Failed to retrieve SAP data for FabNo: {fabNo}");
        return [];
    }

    if (sapData.OpsEquiData?.EquiData?.ItemList == null ||
        sapData.OpsEquiData?.OpsData?.ItemList == null)
    {
        NLogging.Instance.Error($"SAP data for FabNo: {fabNo} is incomplete or missing.");
        return [];
    }
    ...
}
```

Auch hier ist der Fokus klar erkennbar: Fehler sollen sichtbar sein, die Anwendung soll aber nach Möglichkeit weiterlaufen. Das Ergebnis ist dann kein vollständiger SAP-Vergleich, sondern eine leere Vergleichsmenge. Für den Benutzer ist das wesentlich besser als ein vollständiger Abbruch der gesamten Seite.

Dasselbe Muster findet sich auch bei der Konfiguration. In `Settings.LoadXml` wird geprüft, ob die XML-Datei vorhanden ist. Fehler beim Laden werden geloggt. In `FileBlacklist.Load` wiederum führen ungültige Regex-Muster nicht zu einem Abbruch der Anwendung, sondern werden lediglich protokolliert und übersprungen. Dadurch bleibt das System auch dann lauffähig, wenn ein einzelner Konfigurationseintrag fehlerhaft ist.

```
private static List<Regex> Load()
{
    List<Regex> result = [];
    foreach (string item in Config.Settings.Instance.Blacklist)
    {
        try
        {
            result.Add(new Regex(item, RegexOptions.IgnoreCase));
        }
        catch (ArgumentException)
        {
            NLogging.Instance.Error($"Invalid regex pattern in blacklist: {item}");
        }
    }
    return result;
}
```

Selbst beim Parsen von `.fcl`-Dateien ist die Verarbeitung robust ausgelegt. Tritt in einer Zeile ein Fehler auf, wird dieser geloggt. Die übrigen Zeilen können dennoch weiterverarbeitet werden. Das verhindert, dass eine einzelne inkonsistente XML-Datei den kompletten Dateistrukturvergleich unbrauchbar macht.

Stabile Rückgabeformate statt Abbruchlogik

Ein wichtiger Aspekt der Fehlerbehandlung liegt nicht nur im Logging, sondern auch in der Entscheidung, welche Rückgabeformate bei Fehlern verwendet werden. Der Strukturvergleich liefert

bei ungültigen Projektpfaden etwa kein `null`, sondern ein leeres `FileStructureComparison`-Objekt mit den ursprünglich übergebenen Pfaden. Für die weitere Verarbeitung im Frontend ist das hilfreich, weil die erwartete Struktur der Antwort erhalten bleibt.

Ähnlich verhält sich der Datei-Diff. Existiert eine der beiden Dateien nicht, wird keine Exception geworfen. Stattdessen bleibt die jeweilige Zeilenliste leer. Der Diff kann danach trotzdem berechnet werden. Das Frontend bekommt also ein formal vollständiges `FileComparison`-Objekt und kann den Zustand „nur links vorhanden“ beziehungsweise „nur rechts vorhanden“ über die vorhandenen Statuswerte darstellen.

Dieses Designprinzip erhöht die Robustheit des Systems. Die Oberfläche muss nicht für jede Speziallage eine völlig andere Fehlerstruktur behandeln, sondern kann in den meisten Fällen mit denselben Modellen weiterarbeiten. Die eigentliche Fehlerursache bleibt im Log nachvollziehbar, während die API nach außen möglichst konsistent bleibt.

5.2.4 Herausforderungen

Im Laufe der Backend-Implementierung sind einige Herausforderungen aufgefallen, die über das reine Schreiben von Code hinausgehen. Diese Punkte sind wichtig, um zu verstehen, warum bestimmte Lösungen im Backend so umgesetzt wurden.

Uneinheitliche Projektstrukturen und Pfadnormalisierung

Eine Herausforderung ergibt sich aus der Struktur der zu vergleichenden Projekte. Diese liegen nicht immer gleich vor. Ein häufiger Fall ist, dass der eigentliche Projektwurzelordner noch unterhalb eines zusätzlichen Verzeichnisses `harddisk0` liegt. Ohne Anpassung würde das Backend solche Projekte auf Pfadebene unterschiedlich behandeln, obwohl sie inhaltlich gleich sind.

Die Methode `Project.ResolvePath` sorgt dafür, dass diese Unterschiede ausgeglichen werden. Dadurch arbeiten Overview-Vergleich, Strukturvergleich und SAP-Vergleich auf derselben Grundlage. Das zeigt auch, dass Probleme oft schon bei scheinbar einfachen Dingen wie Pfaden beginnen.

Relevante und irrelevante Dateien unterscheiden

Beim Vergleich von Projektordnern kann nicht einfach jede Datei berücksichtigt werden. In den Verzeichnissen liegen oft zusätzliche Dateien wie generierte Artefakte oder Debug-Dateien,

die für den Vergleich nicht relevant sind. Würden diese einbezogen, wäre das Ergebnis schnell unübersichtlich und unnötig groß.

Die Blacklist in `Settings.xml` filtert solche Dateien heraus. Dabei muss sie einerseits flexibel genug sein, um neue Fälle abzudecken, darf andererseits aber keine wichtigen Dateien ausschließen. Die Verwendung von Regex macht das möglich, erfordert aber auch etwas Sorgfalt bei der Pflege.

Zusätzlich spielen die `.fcl`-Dateien eine Rolle. Sie zeigen, welche Dateien in einem Projekt tatsächlich verwendet werden. Dadurch geht es beim Vergleich nicht nur darum, Dateien aufzulisten, sondern sie auch im Kontext zu betrachten.

Performance und Lesbarkeit im Strukturvergleich ausbalancieren

Beim Strukturvergleich mussten zwei Dinge gleichzeitig berücksichtigt werden: gute Performance und ein verständliches Ergebnis für die Oberfläche. Beides lässt sich nicht immer direkt kombinieren.

Die aktuelle Lösung versucht hier einen Mittelweg. Das Traversieren des Dateisystems läuft parallel. Für Dateien werden Hashwerte berechnet, sodass spätere Vergleiche schneller sind. Gleichzeitig werden Platzhalterknoten eingefügt, damit beide Seiten im Vergleich zueinander passen.

Diese Platzhalter sind technisch nicht zwingend notwendig, helfen aber bei der späteren Darstellung. Damit wird deutlich, dass die interne Struktur nicht nur für die Verarbeitung, sondern auch für die Ausgabe gedacht ist.

Einen verständlichen Datei-Diff erzeugen

Auch beim Datei-Diff lag eine Herausforderung darin, ein sinnvolles Ergebnis für den Benutzer zu liefern. Ein einfacher Vergleich „Datei A ist ungleich Datei B“ reicht nicht aus. Ein rein zeilenweiser Vergleich ist oft ebenfalls zu grob.

Die Lösung besteht aus zwei Schritten: zuerst ein Vergleich auf Zeilenebene, danach ein zusätzlicher Vergleich innerhalb der geänderten Zeilen. Dafür wird eine Tokenisierung verwendet.

Zu Beginn wurde auch mit eigenen Vergleichsansätzen experimentiert. Dabei hat sich jedoch schnell gezeigt, dass ein selbst implementierter, stabiler und sinnvoller Zeilenabgleich den Rahmen sprengen würde. Daher wird für den grundlegenden Vergleich eine Library verwendet, die anschließend durch die zusätzliche Tokenisierung ergänzt wird.

Durch die zweite Ebene können Änderungen innerhalb einer Zeile genauer dargestellt werden. Damit die Ausgabe nicht zu kleinteilig wird, werden zusammenhängende Bereiche am Ende wieder zusammengefasst.

Abhängigkeit von SAP und unvollständigen externen Daten

Beim SAP-Vergleich kommt eine weitere Schwierigkeit hinzu: Das Backend ist hier von externen Daten abhängig. Fehlende `FabNumber`, unvollständige Antworten oder Fehler beim Abruf können dazu führen, dass der Vergleich nicht vollständig ist.

Die Implementierung ist deshalb so aufgebaut, dass solche Fälle abgefangen werden. Fehler werden geloggt, der restliche Vergleich läuft aber weiter. Das ist sinnvoll, weil SAP nur ein Teil des Gesamtsystems ist.

Damit zeigt sich auch ein typisches Problem bei Integrationen: Man kann nicht immer von vollständigen oder fehlerfreien Daten ausgehen. Wichtig ist daher, wie stabil das System mit solchen Situationen umgeht.

6 Ergebnis

Im Rahmen der Diplomarbeit wurde eine Anwendung entwickelt, welche den Vergleich zweier Softwareprojekte ermöglicht und Unterschiede übersichtlich in einer grafischen Benutzeroberfläche darstellt. Die Anwendung wurde als Desktop-Applikation auf Basis von Angular und Electron umgesetzt und kommuniziert über eine REST-Schnittstelle mit einem C# ASP.NET Backend.

6.1 Projektauswahl

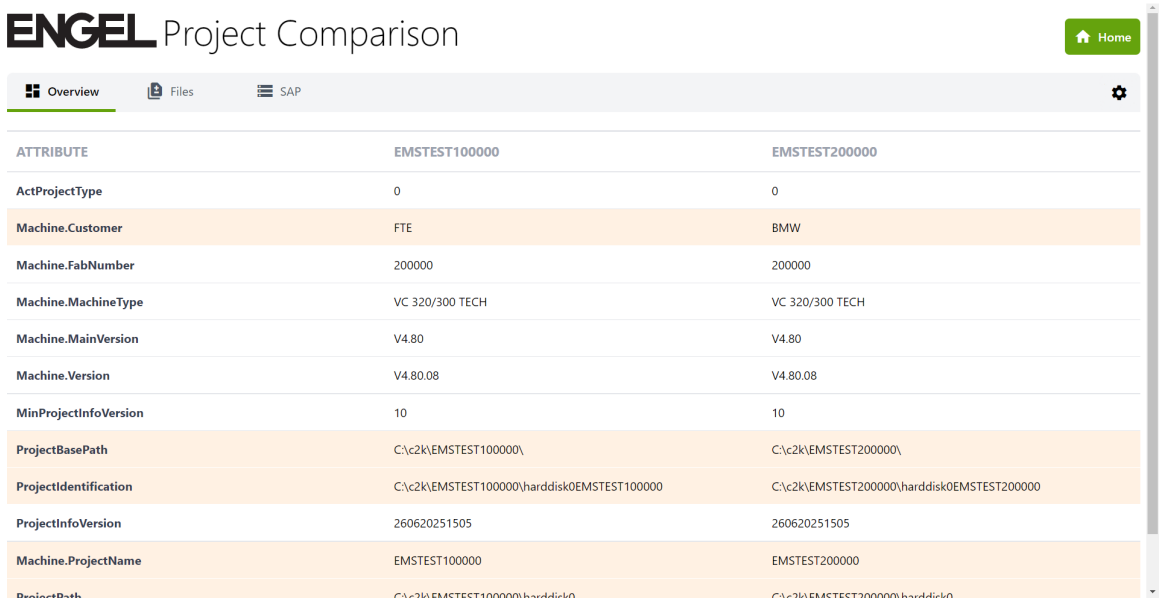
Zu Beginn der Anwendung kann der Benutzer zwei Softwareprojekte auswählen und hochladen, die miteinander verglichen werden sollen. Nach erfolgreicher Auswahl wird der Vergleich gestartet und die Anwendung wechselt zur Vergleichsansicht.

Abbildung 29: Auswahl und Upload der zu vergleichenden Projekte

6.2 Übersichtsansicht

Nach dem Start des Vergleichs wird zunächst eine Übersichtsseite angezeigt, welche allgemeine Informationen zum Vergleich der beiden Projekte bereitstellt. Diese Ansicht dient als Einstiegspunkt und ermöglicht einen schnellen Überblick über die wichtigsten Unterschiede.

Zusätzlich kann der Benutzer über eine Checkbox festlegen, ob ausschließlich Unterschiede angezeigt werden sollen. Dadurch wird die Übersichtlichkeit insbesondere bei größeren Datenmengen verbessert.



ATTRIBUTE	EMSTEST100000	EMSTEST200000
ActProjectType	0	0
Machine.Customer	FTE	BMW
Machine.FabNumber	200000	200000
Machine.MachineType	VC 320/300 TECH	VC 320/300 TECH
Machine.MainVersion	V4.80	V4.80
Machine.Version	V4.80.08	V4.80.08
MinProjectInfoVersion	10	10
ProjectBasePath	C:\c2k\EMSTEST100000\	C:\c2k\EMSTEST200000\
ProjectIdentification	C:\c2k\EMSTEST100000\harddisk0EMSTEST100000	C:\c2k\EMSTEST200000\harddisk0EMSTEST200000
ProjectInfoVersion	260620251505	260620251505
Machine.ProjectName	EMSTEST100000	EMSTEST200000

Abbildung 30: Übersichtsseite des Projektvergleichs

6.3 Baumstrukturansicht (TreeView)

Ein zentraler Bestandteil der Anwendung ist die Darstellung der Projektstruktur in Form einer Baumansicht. Hier werden Ordner und Dateien beider Projekte hierarchisch dargestellt.

Unterschiede werden visuell hervorgehoben, um eine schnelle Orientierung zu ermöglichen. Dabei werden neu hinzugefügte Dateien und Ordner grün dargestellt, während geänderte Elemente orange markiert werden.

Zusätzlich besteht die Möglichkeit, die Zugehörigkeit zu sogenannten FCL-Dateien optional über ein Symbol einzublenden.

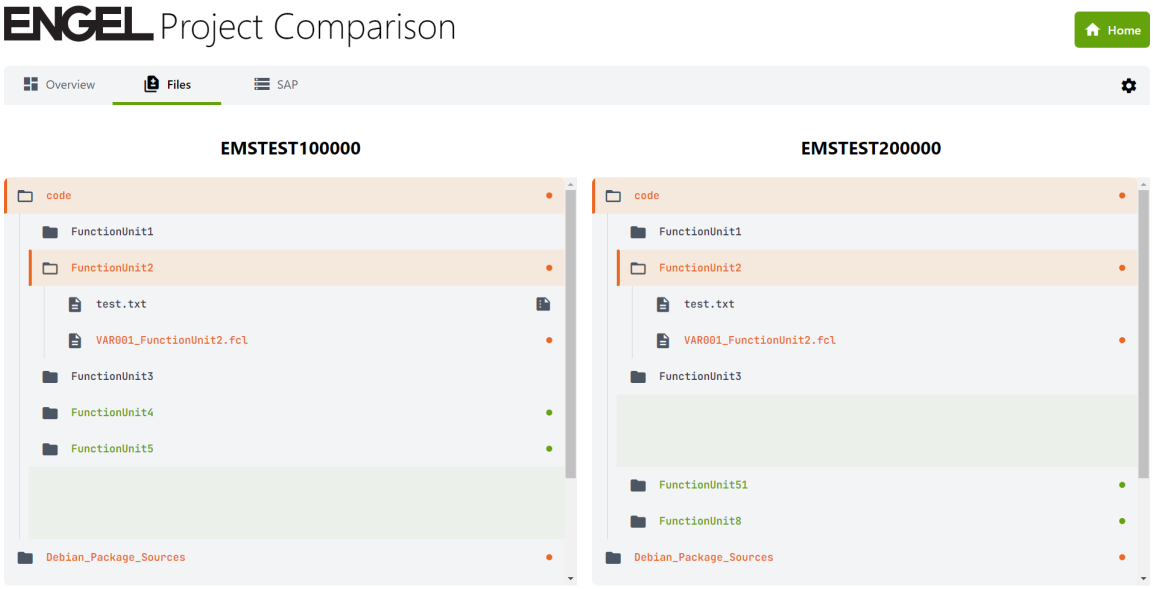


Abbildung 31: Baumstrukturansicht mit hervorgehobenen Unterschieden

6.4 Dateivergleich (FileView)

Durch einen Doppelklick auf eine Datei kann eine Detailansicht geöffnet werden, in der die Unterschiede zwischen den Dateien zeilenweise dargestellt werden. Diese Ansicht ermöglicht eine genaue Analyse von Änderungen innerhalb der Dateien.

Abbildung 32: Ansicht eines Dateivergleichs

6.5 SAP-Ansicht

Neben der Dateiansicht steht ein weiterer Tab zur Verfügung, in dem SAP-spezifische Daten dargestellt werden. Diese ermöglichen einen zusätzlichen Vergleich auf Basis von ERP-relevanten Informationen.

Auch in dieser Ansicht kann der Benutzer über eine Checkbox festlegen, ob ausschließlich Unterschiede angezeigt werden sollen.

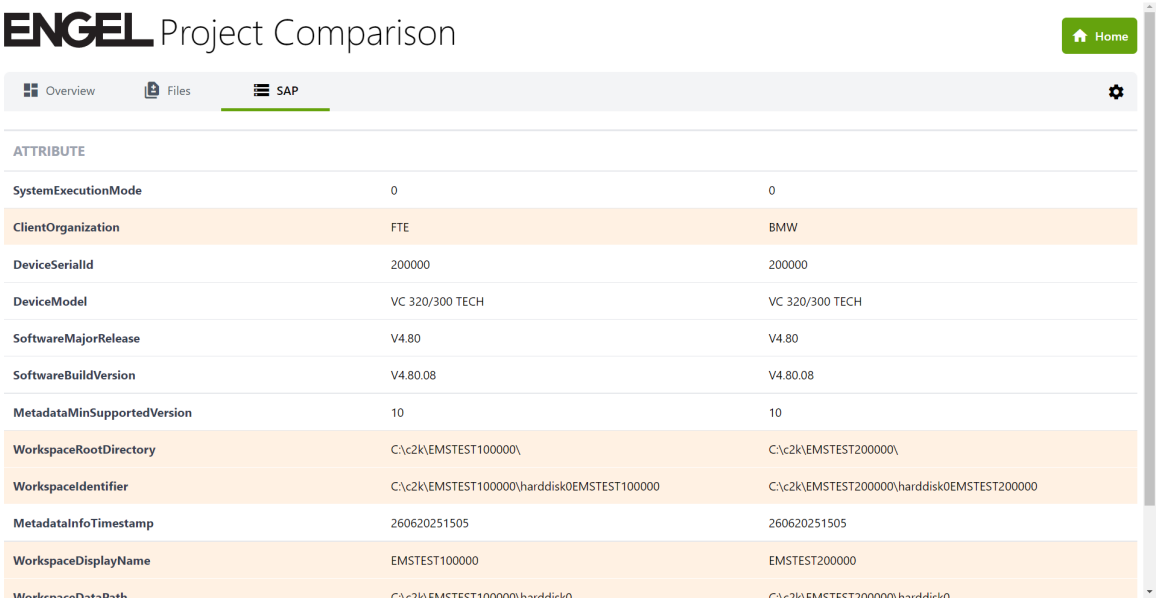


Abbildung 33: Darstellung der SAP-bezogenen Vergleichsdaten

6.6 Backend

Das Backend stellt eine REST-Schnittstelle zur Verfügung, über welche die Webanwendung die benötigten Vergleichsdaten abruft. Die implementierten Endpunkte ermöglichen unter anderem das Starten eines Projektvergleichs sowie das Abrufen detaillierter Dateiunterschiede.

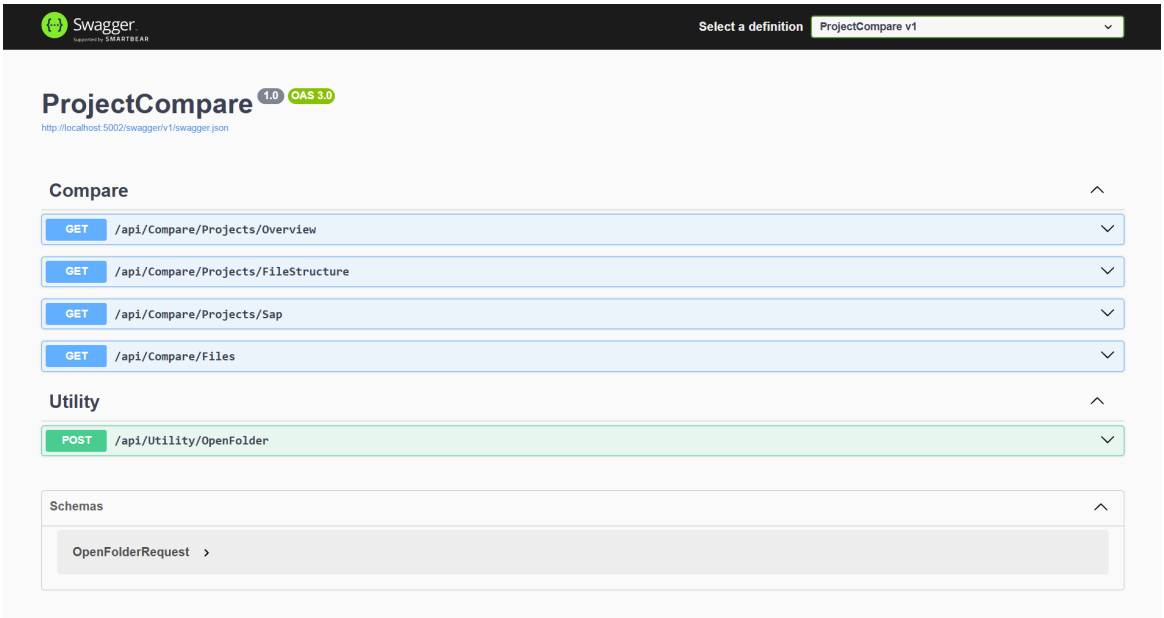


Abbildung 34: Swagger-Dokumentation der REST-Endpunkte

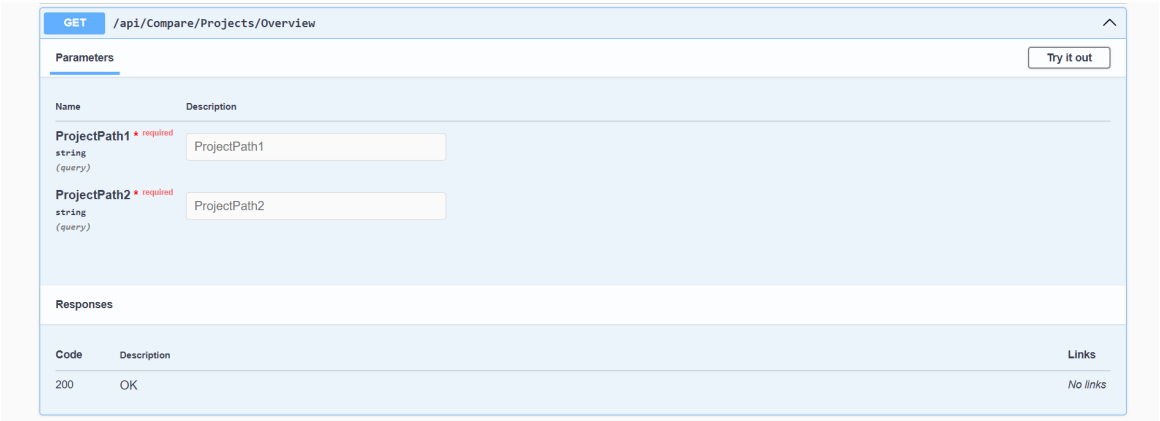


Abbildung 35: Swagger-Detaildokumentation des Overview-Endpunkts

Die Abbildung 34 zeigt die automatisch generierte Swagger-Dokumentation der Backend-Schnittstellen. Hier sind alle verfügbaren Endpunkte sowie deren Parameter und Rückgabewerte ersichtlich (Beispiel siehe Abbildung 35).

Die Verwendung von Swagger erleichtert die Entwicklung und das Testen der Schnittstellen, da die Endpunkte direkt über eine grafische Oberfläche aufgerufen werden können.

7 Resümee

Der strukturierte Vergleich von Softwareprojekten ist in der Praxis eine anspruchsvolle Aufgabe, insbesondere dann, wenn Projekte aus vielen Ordnern, Dateien und zusätzlichen projektspezifischen Informationen bestehen. Ein rein manueller Vergleich ist in solchen Fällen zeitaufwendig, unübersichtlich und fehleranfällig. Im Rahmen dieser Diplomarbeit wurde daher eine Anwendung entwickelt, die diesen Vergleich automatisiert und die Unterschiede übersichtlich darstellt.

Die entwickelte Lösung ermöglicht es, zwei Softwareprojekte auf mehreren Ebenen miteinander zu vergleichen. Neben allgemeinen Projektinformationen werden auch die Dateistruktur, Inhalte einzelner Dateien sowie SAP-bezogene Daten berücksichtigt. Dadurch entsteht ein umfassender Vergleich, der dem Benutzer nicht nur grobe Unterschiede zeigt, sondern auch eine detaillierte Analyse einzelner Änderungen erlaubt.

Das Ergebnis der Arbeit ist die Umsetzung einer praxisnahen Fullstack-Anwendung, bestehend aus einem Angular-Frontend mit Electron und einem C#-Backend mit ASP.NET. Die Kommunikation zwischen den beiden Teilen erfolgt über eine REST-Schnittstelle. Durch die modulare Architektur, die klare Trennung von Oberfläche, Logik und Datenverarbeitung sowie die strukturierte Aufteilung in Komponenten, Services und Controller konnte eine wartbare und erweiterbare Lösung geschaffen werden.

Besonders wichtig war dabei nicht nur die fachliche Korrektheit des Vergleichs, sondern auch die Benutzerfreundlichkeit. Unterschiede werden in der Anwendung visuell hervorgehoben und in verschiedenen Ansichten dargestellt. So erhält der Benutzer zunächst einen Überblick über die wichtigsten Abweichungen und kann anschließend bis auf Datei- und Zeilenebene in die Details einsteigen. Zusätzlich wurde darauf geachtet, dass das System auch bei unvollständigen oder fehlerhaften Daten robust reagiert und stabile Rückgabeformate liefert.

Die Arbeit zeigt außerdem, dass für einen aussagekräftigen Projektvergleich nicht nur Dateiinhalte relevant sind. Auch Metadaten, projektspezifische Konfigurationen, SAP-Informationen sowie die Einordnung von Dateien im Gesamtkontext spielen eine wichtige Rolle. Erst durch die Kombination dieser Informationen entsteht ein Tool, das im Unternehmensumfeld echten praktischen Nutzen bietet.

Insgesamt konnte mit der Diplomarbeit eine funktionale Anwendung entwickelt werden, die den Vergleich von Softwareprojekten deutlich erleichtert und den manuellen Aufwand reduziert. Gleichzeitig wurde eine technische Grundlage geschaffen, auf der zukünftige Erweiterungen und eine weitere Integration in die bestehende Systemlandschaft von ENGEL aufbauen können. Das Projekt zeigt damit, wie moderne Webtechnologien und eine klar strukturierte Backend-Architektur eingesetzt werden können, um ein konkretes Problem aus dem Unternehmensalltag zielgerichtet zu lösen.

8 Aufgabenverteilung

Im Folgenden wird festgelegt, welcher Schüler dieser Arbeit welches inhaltliche Thema verfasst hat.

8.1 Lukas Langeder

Inhaltsverzeichnis Langeder

1 Einleitung **1**

1.3 Zielsetzung 2

1.4 Projektinhalt 2

1.5 Projektumfeld 3

2 Grundlagen und Methoden **5**

2.1 Grundlegende Fachbegriffe 5

2.1.1 DOM 5

2.1.2 Framework 6

2.1.3 Komponenten 6

2.1.4 Services 7

2.2 Verwendete Technologien 11

2.2.1 Frontend 11

2.3 Verwendete Entwicklungssysteme 16

2.3.1 Visual Studio Code 16

2.3.3 Figma 18

2.3.4 Chrome DevTools 18

2.4 Bibliotheken und Plug-Ins 21

2.4.1 Frontend 21

3 Planung und Realisierung 28

3.1 Projektorganisation 28

3.2 Meilensteine 28

4 Gesamtarchitektur 30

4.1 Funktionale und Nicht-funktionale Anforderungen 30

4.2 Technischer Überblick 32

4.2.1 Frontend 32

4.4 Projektstruktur 36

4.4.1 Frontend 36

5 Implementierung 43

5.1 Webanwendung 43

6 Ergebnis 84

6.1 Projektauswahl 84

6.2 Übersichtsansicht 84

6.3 Baumstrukturansicht (TreeView) 85

6.4 Dateivergleich (FileView) 86

6.5 SAP-Ansicht 87

8.2 Andreas Prinz

Inhaltsverzeichnis Prinz

1 Einleitung **1**

1.1 Motivation 1

1.2 Ausgangslage 1

2 Grundlagen und Methoden **5**

2.1 Grundlegende Fachbegriffe 5

2.1.5 DTOs 7

2.1.6 Unit-Tests 8

2.1.7 RESTful 9

2.1.8 Interfaces 10

2.2 Verwendete Technologien 11

2.2.2 Backend 13

2.2.3 SAP 15

2.3 Verwendete Entwicklungssysteme 16

2.3.2 Visual Studio 16

2.3.5 Git 19

2.3.6 Postman 20

2.4 Bibliotheken und Plug-Ins 21

2.4.2 Backend 24

4 Gesamtarchitektur **30**

4.2 Technischer Überblick 32

4.2.2 Backend 32

4.2.3 ERP-System 33

4.3 Softwarearchitektur 34

4.4 Projektstruktur 36

4.4.2 Backend 39

5 Implementierung **43**

5.2 Backend 59

6 Ergebnis **84**

6.6 Backend 88

7 Resümee **89**

Glossar

API Application Programming Interface

CLI Command Line Interface

CLR Common Language Runtime

CORS Cross-Origin Resource Sharing

CSS Cascading Style Sheets

DOM Document Object Model

DTO Data Transfer Object

ERP Enterprise Resource Planning

FCL Function Control List

Git Global Information Tracker

HANA HochleistungsANalyseAnwendung

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IBM International Business Machines Corporation

IDE Integrated Development Environment

IIS Internet Information Services

JIT Just-in-Time

JSON JavaScript Object Notation

MVC Model-View-Controller

NPM Node Package Manager

Regex Regular Expression

REST Representational State Transfer

RxJS Reactive Extensions for JavaScript

SAP Systeme, Anwendungen und Produkte in der Datenverarbeitung

SDK Software Development Kit

SHA Secure Hash Algorithm

SPA Single Page Application

UI User Interface

UML Unified Modeling Language

URL Uniform Resource Locator

v. l. n. r. von links nach rechts

XML Extensible Markup Language

Literaturverzeichnis

[1] ENGEL Austria GmbH. (2026) Hersteller von Spritzgieß-Lösungen. letzter Zugriff am 25.03.2026. Online verfügbar: https://www.engelglobal.com/de/at/unternehmen/engel-spritzgiessen

[2] MDN Web Docs, „Document Object Model (DOM) - Web API," 2026, letzter Zugriff am 17.01.2026. Online verfügbar: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

[3] MDN Web Docs, „Client-side web frameworks," 2024, letzter Zugriff am 17.03.2026. Online verfügbar: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks

[4] HTW Berlin. (2026) Angular - WebTech. letzter Zugriff am 18.03.2026. Online verfügbar: https://freiheit.f4.htw-berlin.de/webtech/angular/

[5] Angular Team. (2026) Anatomy of components. letzter Zugriff am 18.03.2026. Online verfügbar: https://angular.dev/guide/components

[6] Angular Team. (2026) Components. letzter Zugriff am 18.03.2026. Online verfügbar: https://angular.dev/essentials/components

[7] O. Tonka, „What Is Separation Of Concerns?" 2023, letzter Zugriff am 17.03.2026. Online verfügbar: https://medium.com/@okay.tonka/what-is-separation-of-concern-b6715b2e0f75

[8] Angular, „Dependency injection," 2026, letzter Zugriff am 17.01.2026. Online verfügbar: https://angular.dev/guide/di

[9] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[10] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.

[11] Microsoft, „Web API design best practices," 2023, letzter Zugriff am 17.03.2026. Online verfügbar: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design

[12] Microsoft, „Create a web API with ASP.NET Core - Prevent over-posting," 2024, letzter Zugriff am 17.03.2026. Online verfügbar: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api#prevent-over-posting

[13] Microsoft, „Model validation in ASP.NET Core MVC and Web API," 2024, letzter Zugriff am 17.03.2026. Online verfügbar: https://learn.microsoft.com/en-us/aspnet/core/mvc/models/validation

[14] Microsoft, „Testing in .NET," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://learn.microsoft.com/en-us/dotnet/core/testing

[15] Microsoft, „Unit testing best practices for .NET," 2025, letzter Zugriff am 17.03.2026. Online verfügbar: https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices

[16] R. T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures," Dissertation, University of California, Irvine, 2000.

[17] Microsoft, „Microsoft REST API Guidelines," 2025, letzter Zugriff am 17.03.2026. Online verfügbar: https://github.com/microsoft/api-guidelines/blob/vNext/graph/GuidelinesGraph.md

[18] L. Richardson, M. Amundsen, und S. Ruby, *RESTful Web APIs: Services for a Changing World*. O'Reilly Media, 2013.

[19] E. Gamma, R. Helm *et al.*, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[20] Microsoft, „.NET Dependency Injection," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://learn.microsoft.com/de-de/dotnet/core/extensions/dependency-injection/overview

[21] Wikimedia, „Node.js logo," 2016, letzter Zugriff am 17.03.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:Node.js_logo.svg

[22] Node.js Foundation, „Node.js Documentation," 2024, letzter Zugriff am 17.03.2026. Online verfügbar: https://nodejs.org/en/about

[23] Wikimedia, „Npm logo," 2014, letzter Zugriff am 17.03.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:Npm-logo.svg

[24] npm Docs, „About npm," 2025, letzter Zugriff am 17.03.2026. Online verfügbar: https://docs.npmjs.com/about-npm

[25] Wikimedia, „Angular gradient logo (2023)," 2023, letzter Zugriff am 17.03.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:Angular_gradient_logo.png

[26] Angular Team, Google, „Angular Documentation," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://angular.dev

[27] Angular Team, Google, „Routing Overview," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://angular.dev/guide/routing

[28] Angular Team, Google, „Angular CLI Overview," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://angular.dev/tools/cli

[29] .NET, „.NET Logo," 2020, letzter Zugriff am 11.03.2026. Online verfügbar: https://github.com/dotnet/brand/blob/main/logo/dotnet-logo.png

[30] .NET, „.NET Downloadseite," 2026, letzter Zugriff am 12.03.2026. Online verfügbar: https://dotnet.microsoft.com/en-us/download

[31] SoftTeco Team, „A Brief History of .NET Framework," 2024, letzter Zugriff am 17.03.2026. Online verfügbar: https://softteco.com/blog/a-brief-history-of-net-framework

[32] M. Bamburic, „.NET Framework is dead – long live .NET 5," 2019, letzter Zugriff am 12.03.2026. Online verfügbar: https://betanews.com/article/future-of-dotnet/

[33] Microsoft, „Überblick über C#," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://learn.microsoft.com/de-de/dotnet/csharp/tour-of-csharp/overview

[34] Microsoft, „Asynchrone Programmierung mit async und await," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://learn.microsoft.com/de-de/dotnet/csharp/asynchronous-programming/

[35] Microsoft, „Intermediate Language & execution," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://learn.microsoft.com/en-us/dotnet/standard/managed-code#intermediate-language--execution

[36] Microsoft, „Common Language Runtime (CLR) overview," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://learn.microsoft.com/en-us/dotnet/standard/clr

[37] R. Lander, „Announcing .NET Core 1.0," 2016, letzter Zugriff am 17.03.2026. Online verfügbar: https://devblogs.microsoft.com/dotnet/announcing-net-core-1-0/

[38] Microsoft, „What is ASP.NET Core?" 2026, letzter Zugriff am 12.03.2026. Online verfügbar: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core

[39] Wikimedia, „SAP Logo," 2011, letzter Zugriff am 12.03.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:SAP_2011_logo.svg

[40] SAP, „What is SAP?" 2024, letzter Zugriff am 17.03.2026. Online verfügbar: https://www.sap.com/about/what-is-sap.html

[41] CompaniesMarketCap, „Largest Software Companies by Market Capitalization," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://www.companiesmarketcap.com/software/largest-software-companies-by-market-cap/

[42] E. Monk und B. Wagner, *Concepts in Enterprise Resource Planning*. Cengage Learning, 2008.

[43] SAP, „The early years: 1972–1980," 2024, letzter Zugriff am 17.03.2026. Online verfügbar: https://www.sap.com/about/company/history.html

[44] SAP, „HP and SAP Accelerate Journey to SAP S/4HANA on HP Helion Managed Cloud," 2015, letzter Zugriff am 17.03.2026. Online verfügbar: https://www.prnewswire.com/news-releases/hp-and-sap-accelerate-journey-to-sap-s4hana-on-hp-helion-managed-cloud-300077441.html

[45] SAP, „SAP website," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://www.sap.com

[46] Wikimedia, „Visual Studio Code 1.35 icon," 2019, letzter Zugriff am 17.03.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:Visual_Studio_Code_1.35_icon.svg

[47] Microsoft, „Visual Studio Code Documentation," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://code.visualstudio.com/docs

[48] Microsoft, „Visual Studio Logo," 2021, letzter Zugriff am 12.03.2026. Online verfügbar: https://visualstudio.microsoft.com/wp-content/uploads/2021/10/Product-Icon.svg

[49] Microsoft, „Was ist Visual Studio?" 2025, letzter Zugriff am 17.03.2026. Online verfügbar: https://learn.microsoft.com/de-de/visualstudio/get-started/visual-studio-ide

[50] Microsoft, „Übersicht über den Visual Studio Debugger," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://learn.microsoft.com/de-de/visualstudio/debugger/debugger-feature-tour

[51] Microsoft, „Microsoft Announces Visual Studio 97," 1997, letzter Zugriff am 17.03.2026. Online verfügbar: https://news.microsoft.com/source/1997/01/28/microsoft-announces-visual-studio-97-a-comprehensive-suite-of-microsoft-visual-development-tools/

[52] Microsoft, „Visual Studio-Editionen vergleichen," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://visualstudio.microsoft.com/de/vs/compare/

[53] Wikimedia, „Figma logo," 2020, letzter Zugriff am 17.03.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:Figma-logo.svg

[54] G. Blandino, „Figma: Was ist es und wie funktioniert es?" 2023, letzter Zugriff am 17.03.2026. Online verfügbar: https://www.pixartprinting.de/blog/figma-was-es-ist/

[55] C. Rau, „Intro to Chrome DevTools," 2020, letzter Zugriff am 20.02.2026. Online verfügbar: https://www.linkedin.com/pulse/intro-chrome-devtools-cole-rau

[56] Google, „Chrome DevTools Documentation," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://developer.chrome.com/docs/devtools

[57] Chrome Developers. (2026) Elements panel overview. letzter Zugriff am 25.03.2026. Online verfügbar: https://developer.chrome.com/docs/devtools/elements

[58] Chrome Developers. (2026) Console overview. letzter Zugriff am 25.03.2026. Online verfügbar: https://developer.chrome.com/docs/devtools/console

[59] Chrome Developers. (2026) Inspect network activity. letzter Zugriff am 25.03.2026. Online verfügbar: https://developer.chrome.com/docs/devtools/network

[60] Wikimedia, „Git Logo," 2012, letzter Zugriff am 12.03.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:Git-logo.svg

[61] Stack Overflow, „Stack Overflow Developer Survey 2022," 2022, letzter Zugriff am 17.03.2026. Online verfügbar: https://survey.stackoverflow.co/2022/

[62] Git, „GitFaq: Why the 'Git' name?" 2012, letzter Zugriff am 17.03.2026. Online verfügbar: https://archive.kernel.org/oldwiki/git.wiki.kernel.org/index.php/GitFaq.html#Why_the_.27Git.27_name.3F

[63] S. Chacon und B. Straub, „Pro Git (2nd ed.)," 2014, letzter Zugriff am 17.03.2026. Online verfügbar: https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

[64] Apress, „A Short History of Git," 2014, letzter Zugriff am 17.03.2026. Online verfügbar: https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git

[65] L. Torvalds, „Meet the new maintainer," 2005, letzter Zugriff am 17.03.2026. Online verfügbar: https://marc.info/?l=git&m=112243466603239

[66] Git, „Git – Branches in a Nutshell," 2020, letzter Zugriff am 17.03.2026. Online verfügbar: https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

[67] Git, „Git website," 2022, letzter Zugriff am 17.03.2026. Online verfügbar: https://git-scm.com/

[68] Wikimedia, „Postman Logo," 2021, letzter Zugriff am 16.03.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:Postman_(software).png

[69] Postman, „What is Postman?" 2021, letzter Zugriff am 17.03.2026. Online verfügbar: https://blog.postman.com/timeline-postman-journey-to-api-platform/

[70] A. Asthana, „How we built Postman—the product and the company," 2021, letzter Zugriff am 17.03.2026. Online verfügbar: https://blog.postman.com/how-we-built-postman-product-and-company/

[71] Postman, „About Us," 2024, letzter Zugriff am 17.03.2026. Online verfügbar: https://www.postman.com/company/contact-us/

[72] BW Disrupt, „Postman moves its corporate headquarters from Bangalore to US," 2017, letzter Zugriff am 17.03.2026. Online verfügbar: https://www.bwdisrupt.com/article/postman-moves-its-corporate-headquarters-from-bangalore-to-us-114508

[73] S. Keegan, „10 Postman features everyone should know," 2021, letzter Zugriff am 17.03.2026. Online verfügbar: https://blog.postman.com/10-postman-features-everyone-should-know/

[74] ReactiveX, „RxJS Logo," 2024, letzter Zugriff am 15.03.2026. Online verfügbar: https://rxjs.dev/

[75] Angular Team, „Observables in Angular," 2024, letzter Zugriff am 15.03.2026. Online verfügbar: https://angular.dev/guide/observables

[76] ReactiveX, „RxJS," 2024, letzter Zugriff am 15.03.2026. Online verfügbar: https://rxjs.dev/

[77] Wikimedia, „Electron Software Framework Logo," 2019, letzter Zugriff am 05.03.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:Electron_Software_Framework_Logo.svg

[78] Electron Developers, „10 Years of Electron," 2023, letzter Zugriff am 04.03.2026. Online verfügbar: https://www.electronjs.org/blog/10-years-of-electron

[79] Electron Developers, „Electron Documentation – Introduction," 2024, letzter Zugriff am 04.03.2026. Online verfügbar: https://www.electronjs.org/docs/latest

[80] Wikimedia, „Tailwind CSS Logo," 2021, letzter Zugriff am 05.03.2026. Online verfügbar: https://commons.wikimedia.org/wiki/File:Tailwind_CSS_Logo.svg

[81] OfferZen, „The Story of Tailwind CSS feat. Adam Wathan," 2022, letzter Zugriff am 04.03.2026. Online verfügbar: https://www.offerzen.com/blog/adam-wathan-story-tailwind-css

[82] Tailwind Labs, „Tailwind CSS Documentation," 2024, letzter Zugriff am 04.03.2026. Online verfügbar: https://tailwindcss.com/docs

[83] Wikimedia, „Swagger Logo," 2017, letzter Zugriff am 12.03.2026. Online verfügbar: https://commons.wikimedia.org/w/index.php?title=File:Swagger-logo.png

[84] julietrome, „Was ist Swagger? Ein Leitfaden für Anfänger," 2024, letzter Zugriff am 17.03.2026. Online verfügbar: https://julietrome.de/swagger/

[85] M. Ralphson, „A brief history of the OpenAPI Specification," 2018, letzter Zugriff am 17.03.2026. Online verfügbar: https://dev.to/mikeralphson/a-brief-history-of-the-openapi-specification-3g27

[86] R. Suter, „ASP.NET Core-Web-API-Dokumentation mit Swagger/OpenAPI," 2026, letzter Zugriff am 17.03.2026. Online verfügbar: https://learn.microsoft.com/de-de/aspnet/core/tutorials/web-api-help-pages-using-swagger

[87] xUnit, „xUnit Logo," 2016, letzter Zugriff am 12.03.2026. Online verfügbar: https://github.com/xunit/media/blob/main/full-logo.png

[88] C. Woodruff, „How to Test ASP.NET Core Web API," 2018, letzter Zugriff am 17.03.2026. Online verfügbar: https://www.infoq.com/articles/testing-aspnet-core-web-api/

[89] NLog, „NLog Logo," 2013, letzter Zugriff am 14.03.2026. Online verfügbar: https://github.com/NLog/NLog.github.io/blob/master/images/NLog.png

[90] NLog, „NLog Website," 2026, letzter Zugriff am 14.03.2026. Online verfügbar: https://nlog-project.org

[91] NLog, „NLog Github," 2026, letzter Zugriff am 14.03.2026. Online verfügbar: https://github.com/NLog/NLog

[92] E. W. Myers, „An O (ND) difference algorithm and its variations," *Algorithmica*, Vol. 1, Nr. 1, S. 251–266, 1986.

Abbildungsverzeichnis

1 Projektteam: von links nach rechts (v. l. n. r.) Andreas Prinz und Lukas Langeder 3
2 ENGEL Logo . . . 3
3 Node.js Logo . . . 11
4 Node Package Manager (NPM) Logo . . . 11
5 Angular Logo . . . 12
6 .NET Logo . . . 13
7 Systeme, Anwendungen und Produkte in der Datenverarbeitung (SAP) Logo . . . 15
8 VS Code Logo . . . 16
9 Visual Studio Logo . . . 16
10 Figma Logo . . . 18
11 Chrome DevTools Logo . . . 18
12 Global Information Tracker (Git) Logo . . . 19
13 Postman Logo . . . 20
14 Reactive Extensions for JavaScript (RxJS) Logo . . . 21
15 Electron Logo . . . 22
16 TailwindCascading Style Sheets (CSS) Logo . . . 23
17 Swagger Logo . . . 24
18 xUnit Logo . . . 25
19 NLog Logo . . . 25
20 Vereinfachter Edit-Graph beim Vergleich zweier Sequenzen . . . 27
21 Softwarearchitektur . . . 34
22 Projektstruktur Frontend . . . 36
23 Projektstruktur Backend . . . 39
24 Overview Comparison Flowchart . . . 66
25 File Structure Comparison Flowchart . . . 71
26 SAP Comparison Flowchart . . . 73
27 File Comparison Flowchart . . . 76
28 Open Folder Flowchart . . . 77
29 Auswahl und Upload der zu vergleichenden Projekte . . . 84
30 Übersichtsseite des Projektvergleichs . . . 85
31 Baumstrukturansicht mit hervorgehobenen Unterschieden . . . 86
32 Ansicht eines Dateivergleichs . . . 86
33 Darstellung der SAP-bezogenen Vergleichsdaten . . . 87
34 Swagger-Dokumentation der Representational State Transfer (REST)-Endpunkte 88
35 Swagger-Detaildokumentation des Overview-Endpunkts . . . 88

Anhang

A API-Dokumentation

Die API-Dokumentation beschreibt die REST-API des Backend-Systems, einschließlich aller verfügbaren Endpunkte sowie deren Request- und Response-Strukturen.

Project Comparison - API Documentation

Endpoints

Compare Endpoints

Project Endpoints

Overview

Method: GET
Path: */api/Compare/Projects/Overview*
Request: **CompareProjectsRequest**
Response: **List**

FileStructure

Method: GET
Path: */api/Compare/Projects/FileStructure*
Request: **CompareProjectsRequest**
Response: **FileStructureComparison**

SAP

Method: GET
Path: */api/Compare/Projects/Sap*
Request: **CompareProjectsRequest**
Response: **List**

File Endpoint

Method: GET
Path: */api/Compare/Files*
Request: **CompareFilesRequest**
Response: **FileComparison**

Utility Endpoints

OpenFolder Endpoint

Method: POST
Path: */api/Utility/OpenFolder*
Request: **OpenFolderRequest**
Response: **Status Code: 200 (success) or 400 (fail)**

Types

CompareProjectsRequest

```
namespace ProjectCompare.Dtos
{
    /// <summary>
    /// Represents a request to compare two project file paths.
    /// The project paths are expected to be provided in the HTTP request body.
    /// </summary>
    public class CompareProjectsRequest
    {
        /// <summary>
        /// The path to the first project file to be compared.
        /// </summary>
        /// <value>
        /// A <see cref="string"/> representing the path to
        /// the first project file.
        /// </value>
        public required string ProjectPath1 { get; set; }

        /// <summary>
        /// The path to the second project file to be compared.
        /// </summary>
        /// <value>
        /// A <see cref="string"/> representing the path to
        /// the second project file.
        /// </value>
        public required string ProjectPath2 { get; set; }
    }
}
```

TableDifference

```
namespace ProjectCompare.Models
{
    /// <summary>
    /// Represents a difference between two projects in a specific table entry.
    /// </summary>
    public class TableDifference
    {
        /// <summary>
        /// The name or identifier of the table entry being compared.
        /// </summary>
        /// <value>
        /// A string representing the name or identifier of the table entry.
        /// </value>
        public required string Name { get; set; } = string.Empty;

        /// <summary>
        /// The value of the entry in the first project.
        /// </summary>
        /// <value>
        /// A string representing the value from the first project.
```

```
        /// </value>
        public required string Project1Value { get; set; }

        /// <summary>
        /// The value of the entry in the second project.
        /// </summary>
        /// <value>
        /// A string representing the value from the second project.
        /// </value>
        public required string Project2Value { get; set; }

        /// <summary>
        /// Indicates whether the values in the two projects differ.
        /// </summary>
        /// <value>
        /// True if the values differ; otherwise, false.
        /// </value>
        public required bool IsDifferent { get; set; }
    }
}
```

FileStructureComparison

```
namespace ProjectCompare.Models
{
    /// <summary>
    /// Represents the result of comparing two projects,
    /// including differences in tables, project file structures, and SAP Data.
    /// </summary>
    public class FileStructureComparison
    {
        /// <summary>
        /// The absolute file system path of the first project.
        /// </summary>
        /// <value>
        /// A string representing the full path to the first project's root directory.
        /// </value>
        public required string Project1AbsolutePath { get; set; }

        /// <summary>
        /// The file structure and details of the first project.
        /// </summary>
        /// <value>
        /// A list of <see cref="FileNodeResponse"/> objects representing
        /// the first project's files.
        /// </value>
        public required List<FileNodeResponse> Project1Files { get; set; }

        /// <summary>
        /// The absolute file system path of the second project.
        /// </summary>
        /// <value>
        /// A string representing the full path to the second project's root directory.
        /// </value>
        public required string Project2AbsolutePath { get; set; }
```

```
        /// <summary>
        /// The file structure and details of the second project.
        /// </summary>
        /// <value>
        /// A list of <see cref="FileNodeResponse"/> objects representing
        /// the second project's files.
        /// </value>
        public required List<FileNodeResponse> Project2Files { get; set; }
    }
}
```

FileNodeResponse

```
namespace ProjectCompare.Models
{
    /// <summary>
    /// Represents a node in a project file tree used in responses,
    /// excluding sensitive or unnecessary Data like file hash.
    /// </summary>
    public class FileNodeResponse
    {
        /// <summary>
        /// The name of the file or directory.
        /// </summary>
        /// <value>
        /// A string representing the name of the file or directory.
        /// </value>
        public required string Name { get; set; } = string.Empty;

        /// <summary>
        /// Indicates whether this node represents a directory (true) or a file (false).
        /// </summary>
        /// <value>
        /// True if the node is a directory; otherwise, false.
        /// </value>
        public required bool IsDirectory { get; set; }

        /// <summary>
        /// Indicates whether this file is referenced in any parsed FCL file.
        /// </summary>
        /// <value>
        /// True if the file path was found in one of the FCL files; otherwise, false.
        /// Always false for directories.
        /// </value>
        public required bool InFcl { get; set; }

        /// <summary>
        /// The relative path of the file or directory from the project root.
        /// Useful for identifying the node's location within the structure.
        /// </summary>
        /// <value>
        /// A string representing the path relative to the project root,
        /// </value>
        public required string RelativePath { get; set; } = string.Empty;
```

```
        /// <summary>
        /// The status of this node indicating differences or similarity
        /// when comparing projects.
        /// </summary>
        /// <value>
        /// A <see cref="FileNodeStatus"/> value representing the
        /// comparison status of the node.
        /// </value>
        public required FileNodeStatus Status { get; set; }

        /// <summary>
        /// The child nodes contained within this directory.
        /// Null or empty if this node is a file or has no children.
        /// </summary>
        /// <value>
        /// A list of child <see cref="FileNodeResponse"/> objects,
        /// or null if there are none.
        /// </value>
        public List<FileNodeResponse>? Children { get; set; }
    }
}
```

FileNodeStatus

```
namespace ProjectCompare.Models
{
    /// <summary>
    /// Represents the status of a file or directory node in a project comparison,
    /// indicating how the node differs between two projects.
    /// </summary>
    public enum FileNodeStatus
    {
        /// <summary>
        /// The node exists only in one of the compared projects.
        /// </summary>
        Only,

        /// <summary>
        /// The node is missing in one of the compared projects.
        /// </summary>
        Missing,

        /// <summary>
        /// The node exists in both projects but with differences.
        /// </summary>
        Different,

        /// <summary>
        /// The node is identical in both compared projects.
        /// </summary>
        Same
    }
}
```

CompareFilesRequest

```
namespace ProjectCompare.Dtos
{
    /// <summary>
    /// Represents a request to compare two individual file paths.
    /// The file paths are expected to be provided in the HTTP request body.
    /// </summary>
    public class CompareFilesRequest
    {
        /// <summary>
        /// The path to the first file to be compared.
        /// </summary>
        /// <value>
        /// A <see cref="string"/> representing the path to the first file.
        /// </value>
        public required string FilePath1 { get; set; }

        /// <summary>
        /// The path to the second file to be compared.
        /// </summary>
        /// <value>
        /// A <see cref="string"/> representing the path to the second file.
        /// </value>
        public required string FilePath2 { get; set; }
    }
}
```

FileComparison

```
namespace ProjectCompare.Models
{
    /// <summary>
    /// Represents the result of comparing two files, containing
    /// the line-by-line differences for each file.
    /// </summary>
    public class FileComparison
    {
        /// <summary>
        /// The list of lines from the first file, including diff metadata.
        /// </summary>
        /// <value>
        /// A list of <see cref="DiffLine"/> objects representing
        /// the first file's lines with diff information.
        /// </value>
        public required List<DiffLine> File1Lines { get; set; }

        /// <summary>
        /// The list of lines from the second file, including diff metadata.
        /// </summary>
        /// <value>
        /// A list of <see cref="DiffLine"/> objects representing
        /// the second file's lines with diff information.
        /// </value>
        public required List<DiffLine> File2Lines { get; set; }
```

```
    }
}
```

DiffLine

```
namespace ProjectCompare.Models
{
    /// <summary>
    /// Represents a single line in a diff result,
    /// including its line number, the content with marked differences,
    /// and the status indicating the type of difference.
    /// </summary>
    public class DiffLine
    {
        /// <summary>
        /// The line number of this line in the original file.
        /// </summary>
        /// <value>
        /// A string representing the line number in the original file.
        /// </value>
        public required string LineNumber { get; set; }

        /// <summary>
        /// The content segments of this line, with each segment indicating if it
        /// is marked as changed.
        /// </summary>
        /// <value>
        /// A list of <see cref="DiffContent"/> segments representing the
        /// line's content and change markers.
        /// </value>
        public required List<DiffContent> Content { get; set; }

        /// <summary>
        /// The status of the line indicating whether it is unchanged,
        /// added, removed, or Modified.
        /// </summary>
        /// <value>
        /// A <see cref="DiffLineStatus"/> value representing
        /// the type of difference for this line.
        /// </value>
        public required DiffLineStatus Status { get; set; }
    }
}
```

DiffContent

```
namespace ProjectCompare.Models
{
    /// <summary>
    /// Represents a piece of content in a diff line, including
    /// the actual text and whether it is marked as different or changed.
    /// </summary>
    public class DiffContent
    {
```

```
        /// <summary>
        /// The textual content of the diff segment.
        /// </summary>
        /// <value>
        /// A string representing the content of the diff segment.
        /// </value>
        public required string Content { get; set; }

        /// <summary>
        /// Indicates whether this content is marked (highlighted).
        /// </summary>
        /// <value>
        /// True if the content is marked as changed; otherwise, false.
        /// </value>
        public required bool IsMarked { get; set; }
    }
}
```

DiffLineStatus

```
namespace ProjectCompare.Models
{
    /// <summary>
    /// Represents the status of a line in a diff comparison,
    /// indicating how the line differs between two files.
    /// </summary>
    public enum DiffLineStatus
    {
        /// <summary>
        /// The line exists only in one of the compared files.
        /// </summary>
        Only,

        /// <summary>
        /// The line is missing in one of the compared files.
        /// </summary>
        Missing,

        /// <summary>
        /// The line exists in both files but with differences.
        /// </summary>
        Different,

        /// <summary>
        /// The line is the same in both compared files.
        /// </summary>
        Same
    }
}
```

OpenFolderRequest

```
namespace ProjectCompare.RequestModels
{
```

```
    /// <summary>
    /// Represents a request to open a folder in the system file explorer.
    /// The folder path is expected to be provided in the HTTP request body.
    /// </summary>
    public class OpenFolderRequest
    {
        /// <summary>
        /// The path to the folder to be opened.
        /// </summary>
        /// <value>
        /// A <see cref="string"/> representing the full directory path.
        /// </value>
        public required string FolderPath { get; set; }
    }
}
```

B Plakat

Das Plakat wurde im Projektentwicklungsunterricht erstellt und dient zur übersichtlichen Präsentation des Projekts und seiner wichtigsten Funktionen.

ENGEL Project Comparison

Ein Tool zum visuellen Vergleich von Softwareprojekten

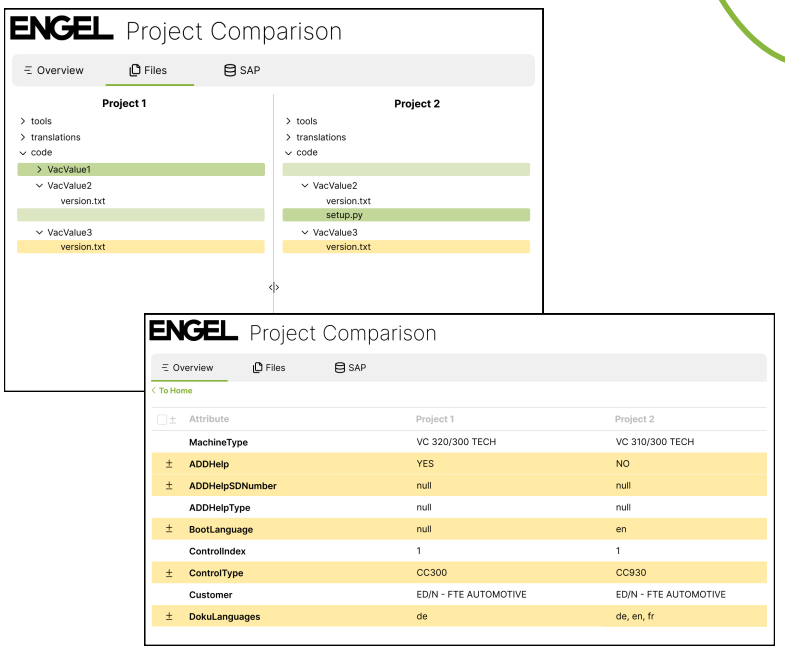
Vergleich von Softwareprojekten auf Dateiebene

Automatische Erkennung von Änderungen, neuen und gelöschten Dateien

Spart Zeit bei Projektanalysen und Reviews

Lukas Langeder
Frontend

Andreas Prinz
Backend



htlperg

C Praxisteildokumentation – Lukas Langeder

Diese Praxisteildokumentation beschreibt die während des Praktikums bei ENGEL durchgeführten Arbeiten im Frontend-Bereich.

Diplomarbeit Frontend

30.06.2025

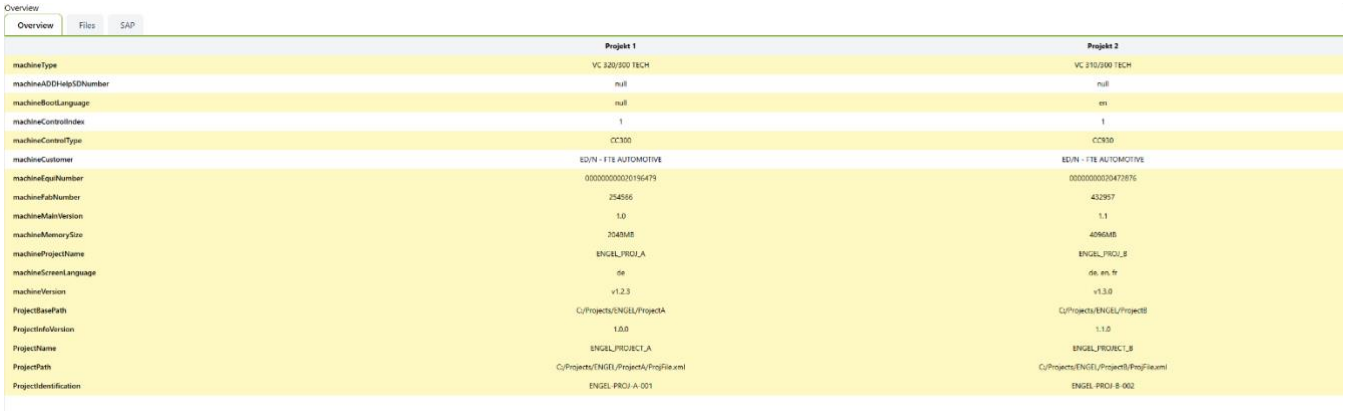
- Einführung ENGEL im Audimax
- Laptops einrichten
- Werksführung
- Besprechung der Aufgabenstellung
- Konzept erstellen
- Figma-Design beginnen

01.07.2025

- Figma-Design fertigstellen & es wurde abgenommen
- Fertiges Frontend (Bereits Angular und Electron installiert) ins eigene Frontend Repo klonen

02.07.2025

- Projekte richtig einrichten, alles Notwendige installieren (tailwind css)
- Vorläufige Tabs erstellen und funktional machen
- Tabellen Component erstellen



	Projekt 1	Projekt 2
machineType	VC 320/300 TECH	VC 210/300 TECH
machineADDHelpSDNumber	null	null
machineBootLanguage	null	en
machineControlIndex	1	1
machineControlType	CC300	CC930
machineCustomer	ED/N - FTE AUTOMOTIVE	ED/N - FTE AUTOMOTIVE
machineEquNumber	000000000020196479	000000000020472876
machineFabNumber	254566	432957
machineMainVersion	1.0	1.1
machineMemorySize	2048MB	4096MB
machineProjectName	ENGEL_PROJ_A	ENGEL_PROJ_B
machineScreenLanguage	de	de, en, fr
machineVersion	v1.2.3	v1.3.0
ProjectBasePath	C:/Projects/ENGEL/ProjectA	C:/Projects/ENGEL/ProjectB
ProjectInfoVersion	1.0.0	1.1.0
ProjectName	ENGEL_PROJECT_A	ENGEL_PROJECT_B
ProjectPath	C:/Projects/ENGEL/ProjectA/ProjFile.xml	C:/Projects/ENGEL/ProjectB/ProjFile.xml
ProjectIdentification	ENGEL-PROJ-A-001	ENGEL-PROJ-B-002

03.07.2025

- Componenten für die eigenen Tabs erstellen
- Eigene Componente für den Tab-Button erstellen und die Logik fürs Tab wechseln implementieren
- Tabellen-Design überarbeiten (Figma)

04.07.2025

- Neues Tabellen-Design implementieren
- Service erstellen, welcher die Daten speichert, einfaches Objekt (Compare ProjectsResponse) + Getter
- Vorläufig mit einen großen Testobjekt arbeiten – im Konstruktor
- Button-Component
    - Icon funktionierte nicht richtig

07.07.2025

- Button Component fixen
- Angular Material installieren
- TreeView implementieren (Component)

08.07.2025

Service wurde so angepasst, dass die Daten aus einer JSON-Datei eingelesen und als Observable bereitgestellt werden

- Mehrere Funktionen zur Verarbeitung des JSONs, insbesondere zur Deserialisierung in die entsprechenden Objekte (z. B. Enum, CompareProjectResponse)
- Probleme bei der Deserialisierung führten dazu, dass einzelne Funktionen nicht korrekt funktioniert haben
- In der TreeView werden Kind-Elemente erst geladen bzw. angezeigt, wenn der Knoten aufgeklappt wird (ngIf mit hasChild)

Verschiebbarer Divider implementiert

09.07.2025

- Daten wurden beim ausführen nicht sofort geladen -> BehaviourSubjects einbauen
- TeamScale Findings beheben
- Statt lucide icons – maticons verwenden
- Beginn Unit-Tests

10.07.2025

- Unittests
- 2 Tickets sind abgeschlossen -> mergen
- Compnente für den File vergleich

11.07.2025

- Objekttypen ändern
- File-compare component beginnen

14.07.2025

- File unterschiede können angezeigt werden und werden farblich markiert
- Probleme: beim scrollen wurde die hintergrundfarbe abgschnitten, es war also nicht durchgängig: min-w-max und mix-w-full lösen das
- Beidseitiges scrollen im component

15.07.2025

- Moveable divider
- Generell components überarbeitet,
    - Tree und file compare components -> logik für den movable divider und enthaltet zwei der kind komponenten
    - Tree und file view enthalten nur den inhalt
- Sync scroll tree und file

16.07.2025

- Größe der panels beim fenster vergrößern war gleich -> verhältnis war nicht gleich
- Sync folder expansion tree view
    - Probleme -> schwer zu lösen

17.07.2025

- Sync Folder expansion Problem gelöst

- TreeState -> inhalt soll erhalten bleiben wenn man ein file wieder schließt
    - Nach einigen kleineren Problemen funktioniert das wieder
- Pull Request

18.07.2025

- Bei Electron funktionierte der File vergleich nicht richtig
    - Gelöst
- FE und BE zusammenbringen

21.07.2025

- Filteransicht ändern
    - Eigenes Dropdown wo man verschiedene Filter auwählen kann
    - Auf allen Tab seiten
- Ein paar State-Bugs fixen
    - Clear-State wurde nicht richtig ausgeführt
- FileNodeDifference: inFcl attribut hinzugefügt, ob eine datei darin enthalten ist,
    - Visuelle Darstellung ob es enthalten mittels togle

22.07.2025

- TreeView komplett umdesignen
- Utility service
- Doubleclick (schreiben) auf einen folder und es öffnet sich der Explorer
- Den name anzeigen statt ‘Project1’ und ‘Project2’

23.07.2025

- File-Endpunkt umbauen das jetzt vom be das richtige file gealden wird
- AbsolutePath nicht mehr aus router auslesen, sondern es wird als response beim file endpunkt zurückgegeben
    - Daher auch die objekte anpassen
- Letzte Design Änderungen durchführen

- Abschlussmeeting
- File-upload component beginnen
- Probleme mit mat-icon, man konnte es die größe nicht verändern -> svg verwenden

D Praxisteildokumentation – Andreas Prinz

Diese Praxisteildokumentation beschreibt die während des Praktikums bei ENGEL durchgeführten Arbeiten im Backend-Bereich.

Dokumentation Diplomarbeit

Backend – Andreas Prinz

Tag 1 (30.06.2025)

- Einführung ENGEL im Audimax
- Laptops einrichten
- Werksführung
- Besprechung der Aufgabenstellung
- Konzept erstellen
- Figma Design erstellen

Tag 2 (01.07.2025)

- Figma Design fertigstellen
- Einrichten/Erstellen der Backend Projekte
    - ProjectCompare (ASP.NET Core-Web-API)
    - ProjectCompare.Tests (xUnit-Testprojekt)
- Teamscale einrichten

Tag 3 (02.07.2025)

- Beginn der Programmierung
- Compare Endpoint → Overview & Filevergleich

Tag 4 (03.07.2025)

- Compare Endpoint → SAP Vergleich
- In Javascript & C# selbstimplementierte Vergleichsexperimente durchführen
- Bibliothek für Filevergleich suchen

Tag 5 (04.07.2025)

- Videos zum Eugen-Meyer-Algorithmus ansehen
  - https://www.youtube.com/watch?v=9n8jI2267MM
  - https://www.youtube.com/watch?v=MGdlTzVyyrE
- Eugen-Meyer Paper durchlesen
  - http://www.xmailserver.org/diff2.pdf
- Diff Bibliothek in C# Projekt einbinden und testen
  - https://github.com/mathertel/Diff
- 1. Funktionierender File-Vergleichs Prototyp (Zeilenbasierend)

Tag 6 (07.07.2025)

- Zeilenbasierenden Prototyp verbessern und fertigstellen
- In Project Comparison Projekt einbinden und JSON Ausgabe implementieren
- HTML Testseite für den Dateivergleich erstellen
- Code Quality improvements
  - Explizite Typen
  - Synchrone Funktionen mit IO Funktionen in Asynchrone Funktionen umwandeln
  - Duplicate Code
  - Kommentare

Tag 7 (08.07.2025)

- Inline Protoyp anlegen und verbessern
- Verschiedene Tokenizers (Word Split, Character Split, ...) testen
  - Jetzt werden Alphanumerische Zeichen (Wörter) von anderen Zeichen getrennt
  - Außer Pfade (/, \\, .) werden als 1 Token behandelt
    - Sonst Probleme beim Vergleich von Pfaden
- Vergleichsdateien erstellen und damit testen
- Inline-Vergleich fertigstellen
- Inline-Vergleich in Hauptprojekt integrieren
- Inline-Vergleich testen

Tag 8 (09.07.2025)

- Sortierung im FileTree anpassen
    - Ordner zuerst, dann Dateien
    - Innerhalb nach Name
- Edge Cases im Code behandeln und Abfragen einfügen um Exceptions zu vermeiden
- Jede Klasse & Funktion kommentieren
- Diff Bibliothek kommentieren, optimieren, verstehen und formatieren
- Teamscale Issues beheben
    - Zu lange Methoden
    - Best Practices (Leerzeilen nach if)
    - Auskommentierter Code
    - string.empty statt “” für Leere Strings
    - ...
- Unit Tests implementieren
- Fine Code Coverage installieren → funktioniert nicht, zu wenig Rechte
- Compare Logic Code dem Betreuer zeigen
    - Pull Request für den Merge in den Master Branch anlegen
    - Feedback/Verbesserungsvorschläge einholen

Tag 9 (10.07.2025)

- Feedback/Verbesserungsvorschläge des Betreuers implementieren
    - Statt ZIP-Projekt nur Pfad übergeben
    - Statt die ganzen Dateien (beim Dateivergleich) nur Pfad übergeben
    - Endpoints aufteilen (Overview, Project Structure, SAP)
    - IEnumerable Sortierung kommentieren
        - Zuerst Ordner dann Dateien
        - Innerhalb Ordner/Dateien nach Name
    - Nur in harddisk0 (hardcoded) springen
        - Vorher wurde der Ordner rekursiv durchsucht:
          Wenn im aktuellen Ordner 0 Dateien und 1 Verzeichnis → Sprung in dieses Verzeichnis (rekursiv)
    - Statt StreamReader (von C#) den LineListProvider (von cc-tools) verwenden
        - cc-tools (selbstimplementierte Libraries von ENGEL)

Tag 10 (11.07.2025)

- Pull Request von Tag 8 mergen
- OpenFolder Endpoint in UtilityService (neu) erstellen
    - Client sendet einen Order-Pfad
    - Öffnet diesen Ordner im Windows Explorer

Tag 11 (14.07.2025)

- Unit Tests für jede Klasse und Funktion erstellen
- Bestehende Unit Tests optimieren
- Unit Tests kommentieren
- Teamscale & Visual Studio Findings bearbeiten
    - IDE Empfehlungen annehmen und Code säubern
    - Methoden auslagern, Best Practices, etc...
- Pull Request für Unit Tests erstellen
- Feedback/Verbesserungsvorschläge des Reviewers implementieren
- Mocking des SAP Datenobjekt versuchen
    - Reales SAP Objekt ist in der CI-Pipeline nicht verfügbar (Zugriff auf SAP-Server nötig)

Tag 12 (15.07.2025)

- Mocking des SAP Datenobjekt versuchen
    - Funktioniert nicht
    - SAP Testmethoden überspringen (mit Skip Parameter)
- Absolute Pfade der beiden Projekte in der CompareFileStructure Response zurückgeben
    - Wenn ein Projekt in harddisk0 verschachtelt ist → Backend löst den Pfad automatisch auf → Frontend benötigt den neuen Pfad für FileDiff
        - Frontend sendet AbsolutePath+RelativePath für FileDiff
- Binary File Blacklist implementieren
    - XML Datei mit Regex-Einträgen: Diese Dateinamen werden ausgeschlossen
    - Pull Request erstellen
- Mit apitest.ps1 Skript die Geschwindigkeit des FileStructure Endpoints testen
    - Verschiedene Hashing Funktionen testen (MD5, SHA1)
    - Fast kein Unterschied zu SHA256 → keine Änderung

Tag 13 (16.07.2025)

- Binary File Blacklist Pull Request
    - LoadXmlToObj statt direkt die XML-Datei einzulesen verwenden
    - Pull Request mergen
- Logging Klasse für NLog implementieren
- Logging Branch Pull Request erstellen

Tag 14 (17.07.2025)

- Logging Branch Pull Request approved → merge
- inFcl Attribut beim Dateistruktur Vergleich einfügen
    - Alle .fcl Dateien werden eingelesen
    - Die enthaltenen Pfade werden in einem HashSet gespeichert
    - Es wird durch beide FileTrees iteriert
        - Wo der relativePath im Fcl-HashSet vorkommt → inFcl = true
- Pull Request für FCL Branch erstellen & mergen
- LINQ-Expressions (Select, Where, Union, etc...) mit ausprogrammierten Methoden ersetzen → später besser wartbar
- Mehr Logpoints einfügen damit ein Laufzeitfehler leichter erkannt werden kann

Tag 15 (18.07.2025)

- Feedback/Verbesserungsvorschläge des Betreuers implementieren
    - Property Methoden auslagern (PropertyUtils Klasse)
    - .Changeable & .Visible Attribute von der Overview entfernen (es wird nur der .Value der ProjectInfo.json angezeigt)
- Restliche Unit-Tests implementieren
- Beim Verbinden des Frontend mit dem Backend helfen
    - launchSettings.json von der .gitignore entfernen → benötigt, da man das Projekt sonst nicht ordnungsgemäß ausführen kann

Tag 16 (21.07.2025)

- Klassendiagramm erstellen
- Flowcharts erstellen (für jeden API Endpoint)
    - ProjectInfo.json Vergleich (/Compare/Projects/Overview)
    - Ordnerstruktur Vergleich (/Compare/Projects/FileStructure)
    - SAP Vergleich (/Compare/Projects/Sap)
    - Inhaltsvergleich zwischen 2 Dateien (/Compare/Files)
    - Pfad im Explorer öffnen (/Utility/OpenFolder)

Tag 17 (22.07.2025)

- Projektpräsentation vorbereiten
- Code nochmals durchgehen, überprüfen und verbessern
    - FileTree.Build() für parallele Verarbeitung umgebaut → Dateistruktur wird nun mithilfe von Multithreading eingelesen und Dateihashes gleichzeitig berechnet.

Tag 18 (23.07.2025)

- File Compare Bug fixen
    - Wenn 1 Datei leer oder nicht existierend → IndexOutOfBound
    - If-Guards und zusätzliche Überprüfungen in Compare() Methode (FileCompare)
- Projekt präsentieren
- GET statt POST verwenden → wo es mehr Sinn macht
    - /Compare/Projects/Overview
    - /Compare/Projects/FileStructure
    - /Compare/Projects/Sap
    - /Compare/Files
- ItemList (Dictionary) statt Property-Reflection beim SAP Vergleich verwenden
    - Einfacher
    - Performanter
- Austrittszettel bearbeiten
    - Alle Daten unserem Betreuer zukommen lassen
    - Unser Projekt auch nach dem Praktikum für uns verfügbar machen
    - Überall abmelden
    - Mitarbeiterkarte entladen/zurückgeben