Entwicklung einer Webanwendung zur automatisierten Ermittlung von Optimierungsmöglichkeiten in Heizungsanlagen

Abschlussarbeit

zur Erlangung des akademischen Grades:

Bachelor of Science (B.Sc.)

an der

Hochschule für Technik und Wirtschaft (HTW) Berlin
Fachbereich 4: Informatik, Kommunikation und Wirtschaft
Studiengang *Angewandte Informatik*

1. Gutachter: Herr Prof. Dr.-Ing. Hendrik Gärtner
2. Gutachter: Herr M.Sc Tobias Pauthner

Eingereicht von Janis Schanbacher

22. Oktober 2021

Gender-Erklärung

Aus Gründen der besseren Lesbarkeit wird in dieser Bachelorarbeit die Sprachform des generischen Maskulinums angewandt. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig verstanden werden soll.

Abstract

Die Effizienzoptimierung von Heizungsanlagen kann einen wichtigen Beitrag zum Umsetzen des Klimaschutzplans 2050 leisten. Die EWUS Effiziente Wärme- und Stromlieferung GmbH hat sich auf diese Anlagenoptimierung spezialisiert und plant regelmäßig Optimierungsmaßnahmen für mehr als 1000 Anlagen. In der vorliegenden Arbeit wird eine Webanwendung entwickelt, welche diese Planung der Optimierungsmaßnahmen der Heizungsanlagen durch Generierung von Textbausteinen, unterstützt. Im Rahmen der Arbeit werden dafür zunächst vier Analysen umgesetzt, welche über eine Benutzeroberfläche ausgelöst werden können. In dieser Benutzeroberfläche können Ingenieure der EWUS GmbH definieren, welche Analysen ausgeführt werden sollen, und die daraus resultierenden Textbausteine bearbeiten und schließlich speichern.

Das Ziel der Arbeit ist es, den Prototyp so zu entwickeln, dass dieser um zusätzliche Analysen erweiterbar ist, und die darin umgesetzte Datenbeschaffung für andere Funktionalitäten wiederverwendet werden kann. Um dies zu ermöglichen, und um die Softwareentwicklung in der EWUS GmbH zu unterstützen, liegt der Fokus dieser Arbeit auf der Auswahl und Umsetzung geeigneter Architekturmuster, Entwurfsmuster und bewährter Richtlinien der Softwareentwicklung. Der Aufbau der Arbeit ist dabei, entsprechend des Vorgehens in Grundlagen, Analyse, Entwurf, Umsetzung und eine Demonstration und Evaluierung unterteilt.

Das Ergebnis dieser Arbeit ist ein lauffähiger Prototyp, welcher die erhobenen Anforderungen umsetzt. Dieser Prototyp wurde im Wesentlichen unter Anwendung einer serviceorientierten Architektur umgesetzt, und verwendet Spring Boot und React als zentrale Technologien. Eines der Teilsysteme des entwickelten Prototyps ist ein Dienst zur Beschaffung der Daten der Heizungsanlagen, welcher für verschiedene Anwendungsfälle wiederverwendet werden kann. Die in der schriftlichen Ausarbeitung erläuterten Entwurfsentscheidungen und Richtlinien zur Entwicklung von einfach verständlichem und anpassbarem Quellcode, sowie eingesetzte Werkzeuge zur Quelltextformatierung, sind weitere Ergebnisse, welche die Softwareentwicklung in der EWUS GmbH unterstützen werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1. Einführung

1.1. Motivation

Der Klimawandel bringt diverse Herausforderungen mit sich, insbesondere die Erfordernis Emissionen einzusparen. Im Energiesektor hat sich die EWUS Effiziente Wärme- und Stromlieferung GmbH darauf spezialisiert, die Effizienz von Heizungsanlagen zu verbessern und somit sowohl Emissionen, als auch Kosten einzusparen[54].
Diesem Unternehmen liegen diverse Messdaten aus Heizungsanlagen und teilweise mittels künstlicher Intelligenz berechnete Kennwerte vor. Da mittlerweile mehr als 1000 Anlagen betreut werden, ist eine manuelle Auswertung der vorliegenden Daten sehr zeitaufwendig, jedoch regelmäßig notwendig. Insbesondere bei der Maßnahmenplanung zur Anlagenoptimierung, jedoch auch bei diversen anderen repetitiven Aufgaben besteht ein großes Potenzial zur Automatisierung.
Bei den Mitarbeitern der EWUS GmbH handelt es sich größtenteils um Ingenieure und einige angehende Softwareentwickler. Aufgrund des Bedarfs einer digitalen Transformation und mangelnder Struktur in der Softwareentwicklung entstanden in der Vergangenheit diverse schwer wartbare und nur minimal wiederverwendbare Anwendungen. Vor diesem Hintergrund sind eine weitsichtig geplante und erweiterbare Softwarearchitektur, sowie eine exemplarische Darstellung des Softwareentwicklungsprozesses und wesentlicher Prinzipien in der Softwareentwicklung sehr hilfreich. Insbesondere kann somit eine Softwareentwicklung gefördert werden, welche wiederverwendbare, erweiterbare und wartbare Ergebnisse erzielt und dabei Entwicklungszeit einspart.

1.2. Zielsetzung

Das zentrale, im Rahmen dieser Bachelorarbeit verfolgte Ziel ist die Entwicklung einer Webanwendung zur automatisierten Ermittlung von Optimierungsmöglichkeiten in Heizungsanlagen. Insbesondere sollen dafür im Rahmen der Bachelorarbeit, basierend auf den vorliegenden Kennwerten und Messdaten die Nutztungsgrade, Anlagengrößen, Rücklauftemperaturen und Temperaturdifferenzen untersucht werden. Auf diese Analysen wird in Abschnitt 3.1.2 eingegangen.
Da zunächst ein Lernprozess bezüglich der Parameter der automatisierten Analysen notwendig ist, soll im Rahmen der Bachelorarbeit eine Benutzeroberfläche erstellt werden, in welcher diese manuell ausgelöst und die Ergebnisse bearbeitet werden

können. Des Weiteren sollen Benutzer konfigurieren können, welche Analysen für die verschiedenen Anlagen ausgelöst werden.
Die Ergebnisse der Analysen liegen in Form von Textbausteinen vor, welche schließlich zur Ticketerstellung in der Anlagenoptimierung verwendet werden können. Die Interaktionen mit den Textbausteinen sollen dabei dokumentiert werden, sodass Wissen zur Anpassung der Analysen und schließlich zur vollständigen Automatisierung der Ticketerstellung gesammelt werden kann.
Im Gegensatz zu der Benutzeroberfläche, welche nur übergangsweise eingesetzt wird, soll der Teil des Systems, in welchem die Datenbeschaffung und Analyse umsetzt wird, als Grundgerüst zur Entwicklung weiterer Automatisierungen dienen. Daher liegt der Fokus dieser Arbeit auf der Konzeption eines Systems, dessen Bestandteile erweiterbar, wiederverwendbar und wartbar sind.

Die Softwareentwicklung in der EWUS GmbH wird bisher wenig angeleitet und größtenteils von Studenten und Ingenieuren durchgeführt. Daher ist ein weiteres Ziel dieser Arbeit, den Softwareentwicklungsprozess und bewährte Richtlinien in der Softwareentwicklung anhand des zu entwickelnden Prototyps zu erläutern.

1.3. Abgrenzung

In dieser Arbeit wird ein Prototyp entwickelt mit dem vorliegende Daten anhand von extern vorgegebenen Bedingungen analysiert werden können. Die Vollständigkeit der verwendeten Datenquellen wird dabei vorausgesetzt. Für fehlende, zur Analyse benötigte, Daten soll lediglich ein entsprechender Hinweis generiert werden.Da die Benutzeroberfläche nur übergangsweise zur frühzeitigen Verwendung der Software und damit zum Ermitteln von Anforderungen an die Analysen verwendet wird, ist das Entwickeln automatisierter Tests von dieser Benutzeroberfläche ausgeschlossen.
Der Fokus liegt auf der Entwicklung einer Grundarchitektur, welche künftig zur vollständigen Automatisierung der Ticketerstellung und zur Umsetzung weiterer Funktionalitäten genutzt werden kann. Ein großer Wert wird dabei auf die Wiederverwendbarkeit der Datenbeschaffung gelegt.

Des weiteren ist zu erwähnen, dass die im Rahmen des Informatikstudiums durchgeführte Bachelorarbeit auf Themen der Software Entwicklung ausgerichtet ist. Somit wird auf die Themen des Energie-Ingenieurwesens nur in dem Umfang eingegangen, wie es für die Analyse des zu entwickelnden Prototyps notwendig ist.
Wie bereits in der Zielsetzung beschrieben, stellt die Arbeit exemplarisch einen Softwareentwicklungsprozess dar. Dieser bezieht sich jedoch lediglich auf den im Rahmen der Bachelorarbeit in Einzelarbeit entwickelten Prototyp. Teambezogene Aspekte der Softwareentwicklung, beispielsweise agiles Vorgehen, sind nicht Teil dieser Arbeit.

1.4. Aufbau der Arbeit

Der Aufbau dieser Arbeit orientiert sich an dem Prozess des Software Engineerings. Dieser ist im Wesentlichen unterteilt in die Analyse des zu entwickelnden Systems, den Entwurf des System, die Umsetzung und eine Evaluierung des Systems. Dabei wird auf das Vorgehen und die Bedeutung dieser Arbeitsschritte, und auf das Ergebnis der jeweiligen Arbeitsschritte eingegangen. Diese Ergebnisse dienen jeweils als Eingabe für den darauffolgenden Arbeitsschritt.
Zu Beginn werden zum Verständnis der Arbeit wichtige Grundlagen geschaffen. Abgeschlossen wird die Arbeit mit einer Demonstration der Ergebnisse, einer Evaluation der Ergebnisse und des Vorgehens und einem Ausblick.

Das Kapitel der Analyse beschäftigt sich mit der Anforderungserhebung und Anforderungsanalyse. Basierend auf den daraus hervorgehenden strukturierten Anforderungen wird das zu entwickelnde System entworfen.
Der Entwurf stellt das zentrale Kapitel der Bachelorarbeit dar. In diesem werden zu Beginn die anzuwendende Architektur und die darin enthaltenen Teilsysteme und deren Abhängigkeiten identifiziert und entworfen. Schließlich wird die Auswahl der einzusetzenden Technologien und die Gestaltung der Benutzeroberfläche geplant.
Daraufhin folgt das Kapitel der Umsetzung, in welchem die Implementierung des entworfenen Systems beschrieben wird. Dabei wird insbesondere auf Maßnahmen eingegangen, welche das Ziel haben, einen einfach zu verstehenden und anpassbaren Quellcode zu entwickeln. Des Weiteren wird in diesem Kapitel auf die Datenbeschaffung und -verwaltung eingegangen, welche die größte Herausforderung in der Umsetzung des Prototypen darstellt. Schließlich wird das Testen der umgesetzten Funktionalitäten thematisiert.
Nach der Beschreibung der Umsetzung wird der entwickelte Prototyp demonstriert und auf die Erfüllung der Anforderungen untersucht. In diesem Kapitel wird auch das Vorgehen evaluiert.
Zum Abschluss der Arbeit erfolgt eine Zusammenfassung mit einem Ausblick hinsichtlich der Weiterentwicklung des umgesetzten Prototyps.

2. Grundlagen

In diesem Kapitel werden einige zum Verständnis der Arbeit notwendige Grundlagen erklärt. Dafür wird zu Beginn auf Grundlagen aus der Softwareentwicklung und schließlich auf einige Fachbegriffe der Anlagenoptimierung eingegangen.

2.1. Client-Server-Architektur

Die Client-Server-Architektur (auch Client-Server-Modell) ist ein Architekturmuster zur Verteilung von Diensten und Aufgaben in einem Netzwerk. In diesem Architekturmuster werden auf die auf einem Server bereitgestellten Dienste von einem oder mehreren Clients konsumiert.

Bei den sogenannten Server und Clients handelt es sich um Rollen. Die Aufgabe des Servers ist es, bestimmte Dienste lokal oder über das Netzwerk bereitzustellen. Der Server nimmt Anfragen der Clients entgegen, verarbeitet sie und sendet die Antwort an die jeweiligen Clients zurück. Wesentlich ist dabei, dass ein Server mehrere Clients bedienen kann.
Der Client hingegen initiiert die Kommunikation in Form von Anfragen an den Server und interpretiert schließlich die erhaltene Antwort. Bei Bedarf können Clients mit mehreren Servern kommunizieren. Häufig stellen Clients Benutzerschnittstellen für die Anwender bereit. Die Kommunikation geschieht dabei unter Anwendung von Netzwerkprotokollen wie HTTP. [36]

Das Client-Server-Modell findet häufig in Webanwendungen Einsatz. Dabei wird die Client-Anwendung, welche im Browser der Benutzer ausgeführt wird, Frontend genannt. Die Anwendung, welche die Rolle des Servers einnimmt wird Backend genannt.[32]

2.2. Monolithische Architektur

Eine monolithische Architektur ist eine Softwarearchitektur bei der alle Komponenten des Systems in einer einzigen untrennbaren Einheit kombiniert werden. Im Gegensatz zum verteilten System gibt es hier keine keine Gliederung in Teilsysteme. Die Komponenten des Systems sind stark gekoppelt, sodass Ausfälle eines Teils des Systems

führen zu einem Ausfall des gesamten Systems. Aufgrund der engen Kopplung sind Monolithen nur schwer skalierbar. Kleine Änderungen des Systems erfordern ein neues Deployment des gesamten Systems, um diese wirksam zu machen. Da alle Komponenten Teil desselben Pakets sind und sich einen gemeinsamen Speicher teilen, ist die Kommunikation zwischen den einzelnen Komponenten schnell und ohne Netzwerkkommunikation möglich. Die monolithische Architektur eignet sich, aufgrund der aufgeführten Eigenschaften, besonders für kleine Anwendungen.[50]

2.3. Serviceorientierte Architektur

Eine serviceorientierte Architektur (SOA, engl. service-oriented architecure) ist im Wesentlichen ein Architekturmuster, welches das System eine Menge von eigenständigen Teilsystemen, sogenannten Services unterteilt [6]. Bei Systemen, welche SOA implementieren handelt es sich also um verteilte Systeme [35]. Ein Service oder Dienst ist dabei eine IT-Repräsentation fachlicher Funktionalität und setzt in der Regel ein Geschäftsziel um [11].

Das Architekturmuster schreibt dabei als Empfehlung gewisse Eigenschaften für die enthaltenen Services vor. Diese werden im Folgenden aufgeführt.
Die Services sollten in sich abgeschlossen sein und eigenständig genutzt werden können. Die Services sollten in einem Netzwerk verfügbar sein und über eine wohldefinierte veröffentlichte Schnittstelle verfügen. Aufgrund dieser Schnittstelle können Implementierungsdetails von den Entitäten, welche die Schnittstelle verwenden, verborgen werden. Services können in unterschiedlichen Programmiersprachen und auf verschiedenen Plattformen realisiert sein, da lediglich der Vertrag (die Schnittstelle) nach außen von Bedeutung ist und die Kommunikation über sprachunabhängige Protokolle geschieht. Die Services sollten in einem Verzeichnis registriert sein. Damit die Abhängigkeiten zwischen den Services minimal ist, sollten diese grobgranular sein. Die benötigte Funktionalität sollte also auf wenige Services verteilt werden.[35]

Wie bereits in den Empfehlungen für die Eigenschaften der Services erwähnt, kommunizieren diese unter Verwendung eines Netzwerkprotokolls miteinander, beispielsweise mit HTTP. Auch die Struktur der über das verwendete Netzwerkprotokoll übertragenen Nachrichten ist einheitlich mittels sprachunabhängiger Protokolle wie REST, JSON oder SOAP vorgeschrieben.
Anstelle der Punkt-zu-Punkt-Verbindungen zwischen Dienstgebern und Dienstnutzern können in SOA auch zwischengeschaltete Vermittler, sogenannte Middleware, zum Einsatz kommen.[35] Da in diesem Projekt die Kommunikation jedoch über Punkt-zu-Punkt-Verbindungen kommuniziert wird, soll an dieser Stelle nicht weiter auf Vermittler eingegangen werden.

Durch Orchestrierung der im System vorhandenen Services niedriger Abstraktionsebene, können diese zu Services auf höheren Abstraktionsebenen kombiniert werden. Somit ermöglicht die serviceorientierte Architektur eine flexible Wiederverwendung der enthaltenen Services für verschiedene Geschäftsziele.[9]

Zum besseren Verständnis serviceorientierter Architekturen ist in Abbildung 2.1 exemplarisch der Aufbau einer solchen Architektur eines Buchhandel dargestellt. Die dargestellte Architektur beinhaltet eine Middleware, den sogenannten Enterprise Service Bus zur Kommunikation. Wie bereits erwähnt wird auf eine derartige Middleware nicht für das in der EWUS GmbH geplante System benötigt.

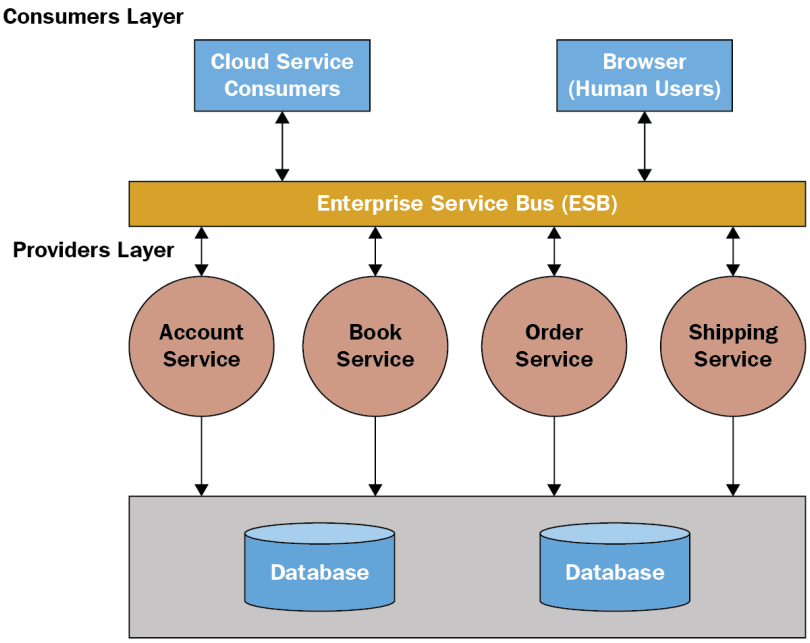

Abbildung 2.1.: *Serviceorientierte Architektur eines Buchhandels; Bildquelle: [45]*

2.4. Microservice Architekturmuster

Im Microservice Architekturmuster, auch Microservices genannt, wird die Funktionalität des Systems auf mehrere, voneinander unabhängige sogenannte Microservices aufgeteilt, welche jeweils in einem eigenen Prozess laufen [4]. Dieses Architekturmuster stellt also das Gegenstück zum traditionellen monolithischen Architekturmuster dar, in welchem das System aus einer einzigen, fest zusammenhängenden Anwendung besteht [33]. Ein Microservice ist ein Dienst, welcher entsprechend der Unix-Philosophie: „Do one thing and do it well“ eine kleine Aufgabe erfüllt. Ein prominentes Merkmal von Microservices ist dabei, dass sie leicht ersetzbar sind und unabhängig voneinander

entwickelt werden können.[2]

Die Kommunikation mit den Microservices geschieht über ein Netzwerk, meist unter Verwendung des HTTP Protokolls. Diese Kommunikation geschieht über standardisierte Schnittstellen, was eine Austauschbarkeit und Anpassbarkeit der einzelnen Services ermöglicht – ohne dabei die Funktionalität der anderen Microservices zu beeinträchtigen [25]. Aufgrund der Kommunikation über standardisierte Schnittstellen, müssen die einzelnen Services nicht auf derselben Programmiersprache oder Technologie basieren [33].

In Abbildung 2.2 wird exemplarisch eine Microservice Architektur dargestellt. Die schwarzen Zylinder symbolisieren dabei Datenbanken, während UI für User Interface (dt. Benutzerschnittstelle) steht.

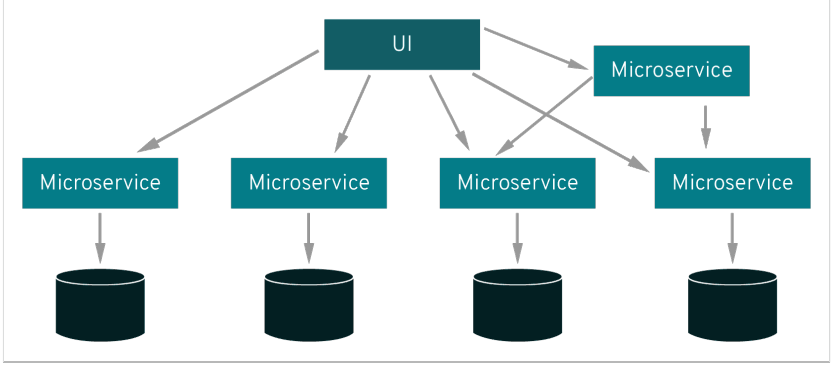

Abbildung 2.2.: *Microservices Architektur; Bildquelle: [24]*

Darüber hinaus können zur horizontalen Skalierung mehrfache Instanzen derselben Services ausgeführt werden. Ist dies der Fall, oder werden die IP Adressen der Microservices dynamisch zugeordnet (beispielsweise im Rahmen der Bereitstellung in Containern), so ist eine sogenannte Service Registry notwendig. Bei der Service Registry handelt es sich um eine Datenbank von laufenden Services und deren Adressen. Die Service Registry kann beispielsweise durch einen Microservice umgesetzt werden, bei welchem sich die Instanzen der anderen Services registrieren. Clients dieser Services können dann deren Adresse anhand des Namens des jeweiligen Services von der Registry erfragen. [48]

2.4.1. Abgrenzung zu SOA

Aufgrund einiger Gemeinsamkeiten mit der serviceorientierten Architektur werden im Folgenden einige Unterschiede aufgeführt.
Zunächst ist auffällig, dass SOA in den 90-er Jahren entstanden ist, während der Begriff

Microservices in 2011 eingeführt wurde. Das lässt den Rückschluss zu, dass Microservices bevorzugt im Zusammenhang mit neueren Technologien eingesetzt werden.
Wie der Name bereits verrät, ist die Aufteilung der Microservices feingranularer als die der Services in SOA. Dieser Unterschied wird in Abbildung 2.3 veranschaulicht, in welcher Monolithen, SOA und Microservices bezüglich Ihrer Granularität verglichen werden.
Ein weiterer Unterschied ist, dass jeder Microservice in der Regel einen eigenen Datenspeicher hat, wohingegen die Services in SOA häufig die gleiche Datenbank verwenden. Dieser Unterschied ist in Abbildung 2.1 und Abbildung 2.2, also den Abbildungen, welche exemplarisch die betrachteten Architekturmuster darstellen, ersichtlich.
Während die Services einer serviceorientierten Architektur meist gemeinsam auf einer Plattform bereitgestellt werden, ist es üblich für Microservices Cloud Plattformen zu verwenden.
Besonders hervorzuheben ist die Zielsetzung der beiden Architekturmuster. Bei SOA liegt der Fokus auf Wiederverwendbarkeit der Services liegt. Das Microservices Architekturmuster ist hingegen auf die Entkopplung der Komponenten und die Möglichkeit diese separat bereitstellen zu können ausgelegt.[1]

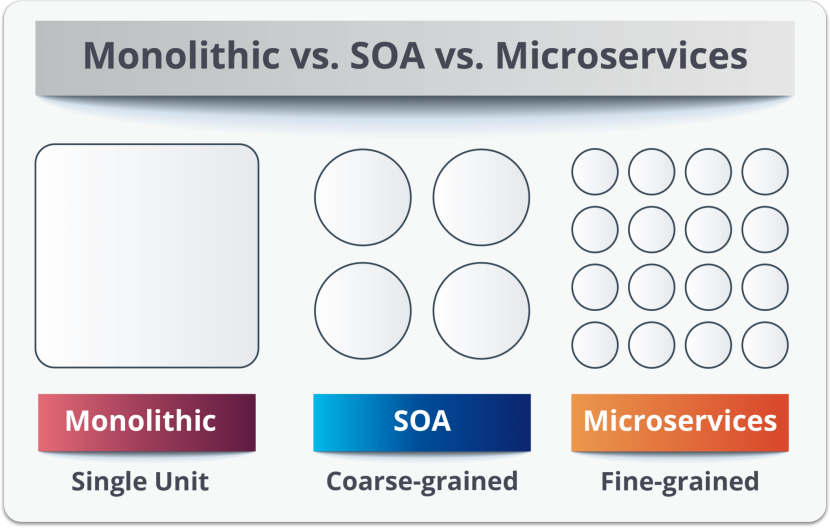

Abbildung 2.3.: *Vergleich der Granularität von Monolithen, SOA und Microservices; Bildquelle: [51]*

2.5. Model-View-Controller

Das Model-View-Controller-Architekturmuster (MVC) ist ein Muster für die Entwicklung interaktiver Systeme. Es kann sowohl als Architekturmuster, also in Hinblick auf die Subsysteme eines Systems und ihrer Beziehungen, als auch als Entwurfsmuster, also in Hinblick auf die Komponenten eines Subsystems und deren Beziehungen untereinander angewandt werden [55]. Es adressiert das Problem, dass die Benutzerschnittstelle

eines Systems häufig Änderungen erfordert und diese daher möglichst austauschbar sein sollten, ohne das restliche System anpassen zu müssen. Im wesentlichen wird das System dabei in drei Teile bzw. Komponenten unterteilt: Model (Verarbeitung/Datenhaltung), View (Präsentationslogik) und Controller (Steuerung von Eingaben bzw. Ausgaben). Diese Aufteilung der Funktionen des Systems in verschiedene Teilsysteme mit entsprechenden Verantwortlichkeiten,ist ein generelles Bestreben der Software-Entwicklung: Trennung der Belange (eng. separation of concerns).[18, S. 377]

Im Model werden die Geschäftsprozesse durchgeführt und die Datenhaltung realisiert. Für das Model ist dabei zu beachten, dass es unabhängig von den Views und Controllern sein sollte, und lediglich von diesen verwendet wird. Somit kann das Model von verschiedenen Controllern und Views verwendet werden.[18, S. 380]
Im Kontext des Unternehmens für dessen Aufgabenstellung die Ergebnisse dieser Bachelorarbeit Anwendung finden werden, ist dies sehr hilfreich, da in diversen Anwendungsfällen Daten der Heiz-Anlagen von den zentralen Webanwendungen Energielenker und Eneffco gelesen und geschrieben werden müssen, diese aber unterschiedlich verwendet werden.

Die View setzt die Darstellung der Daten für den Benutzer um, sie enthält also die Präsentationslogik. Da zur Darstellung der Daten bekannt sein muss, wie diese aufgebaut sind, besteht eine starke einseitige Abhängigkeit der View vom Model. [18, S. 381f]
Teilweise wird die View auch direkt vom Controller aktualisiert, nachdem dieser Benutzereingaben erhalten hat [7].

Der Controller steuert das Model und den Zustand einer View basierend auf den Eingaben des Benutzers. Im Controller werden die Interaktionen des Benutzers interpretiert und entsprechend gegebenenfalls Methoden des Models aufgerufen. Somit fordert der Controller das Model zu einer Anpassung seines Zustandes oder der Daten auf. [18, S. 382f]

Teile des Systems, wie beispielsweise die Analyse-Konfigurationen wurden anhand dieses Architekturmusters erstellt. Darüberhinaus wurde MVC bei der Planung der Verantwortlichkeiten der Microservices in Betracht gezogen, weshalb einer der Microservices im Wesentlichen für das Lesen (und Schreiben) von Daten von Energielenker und Eneffco verantwortlich ist.

2.6. RESTful API

Eine API (von engl. Application programming interface) oder Programmierschnittstelle ist eine Menge von Regeln zum Entwickeln und Integrieren von Software Anwen-

dungen. Sie können als Vertrag zwischen Client und Server bezeichnet werden. Eine Besonderheit ist dabei, dass der Client nicht wissen muss, wie die Bearbeitung seiner Anfrage implementiert ist, sondern lediglich, welches Ergebnis er erwarten kann.[26]

Eine API kann als RESTful API, auch REST API, bezeichnet werden, wenn diese, die im Folgenden aufgeführten, sechs Entwurfsprinzipien des Representational State Transfer (REST) Paradigmas einhält [10]:

- Einheitliche Schnittstelle: Alle Anfragen nach einer Ressource sollten gleich sein, unabhängig davon, woher diese Anfragen kommen. Dieselbe Ressource sollte nur einem Uniform Resource Identifier (URI) zugeordnet sein. Die Ressourcen sollten in selbstbeschreibenden Nachrichten ausgeliefert werden, sodass der Client ausreichende Informationen hat, um diese zu verarbeiten. Ressourcen können von einem Client über dieselbe Repräsentation wie er sie erhält, manipuliert werden können. Nach dem Anfragen einer Ressource sollte der Client in der Lage sein, alle anderen für die Ressource verfügbaren Aktionen über Hyperlinks zu finden.[26]
- Client-Server-Architektur: Die Architektur besteht aus Clients, Servern und Ressourcen, welche ausschließlich über HTTP-Requests miteinander kommunizieren.[26]
- Zustandslosigkeit: Jeder Request muss alle Informationen, die zu dessen Bearbeitung notwendig sind, enthalten. Server Anwendungen dürfen keine auf die Client-Anfragen bezogenen Daten speichern. Es dürfen also keine serverseitigen Sessions erforderlich sein.[10]
- Cachefähigkeit: Wenn möglich, sollten Ressourcen client- oder serverseitig in einem Cache abgelegt werden können, um clientseitig die Performanz und servcerseitig die Skalierbarkeit zu erhöhen.[10]
- Mehrschichtige Systeme: Die Anfragen und Antworten sollten durch mehrere Schichten gehen. REST APIs sollten so designed sein, dass weder Client, noch Server weiß, ob er mit der Endanwendung oder einer Zwischenschicht kommuniziert.[10]
- Code-on-Demand (optional): REST APIs senden normalerweise statische Ressourcen, können aber in bestimmten Fällen auch ausführbaren Code, wie beispielsweise Java applets versenden. In diesen Fällen sollte der Code ausschließlich bei Bedarf ausgeführt werden.[10]

Als Kommunikationsprotokoll kommt dabei das Hypertext Transfer Protocol (HTTP) zum Einsatz [26].
Die mit HTTP übertragenen Nachrichten werden dabei in einem der folgenden Formate versendet: JSON (JavaScript Object Notation), HTML, XLT Python, PHP oder plain text. Am weitesten verbreitet ist dabei jedoch JSON, da dieses Format für Menschen lesbar

und von Programmiersprachen unabhängig ist.[10]

Eine Ressource, das zentrale Element der REST Eigenschaften, kann jegliche Information sein, die benannt werden kann. Dies kann ein Dokument, ein Bild, ein zeitlicher Service oder eine Sammlung von anderen Ressourcen sein [13, S. 88].

Zusammenfassend handelt es sich bei einer RESTful API also um eine Benutzerschnittstelle, welche gemäß der REST Eigenschaften umgesetzt ist.

2.7. Fachbegriffe der Anlagenoptimierung

In diesem Abschnitt werden die zum Verständnis der im Rahmen der Bachelorarbeit durchgeführten Analysen notwendigen Fachbegriffe der Anlagenoptimierung erklärt.

2.7.1. Nutzungsgrad

Der Nutzungsgrad beschreibt welcher Anteil der eingesetzten Energie genutzt werden kann [42]. In Bezug auf Heizungs- und Warmwasseraufbereitungs-Anlagen beschreibt er also das Verhältnis zwischen nutzbarer Wärme und zugeführter Heizenergie.

2.7.2. Heizkessel

Ein Heizkessel dient in Heizungsanlagen der Umsetzung von chemisch gebundener in thermische Energie [41].
Dabei wird durch die Verbrennung des Energieträgers freigesetzte thermische Energie genutzt. Im Gegensatz zu sogenannten Niedertemperaturkesseln wird bei sogenannten Brennwertkesseln zusätzlich die Energie genutzt, welche durch Kondensation des Wasserdampfs freigesetzt wird. In diesem Zusammenhang wird von einem sogenannten Brennwerteffekt gesprochen. Daher kann durch Verwendung von Brennwertkesseln ein etwa 10% höherer Nutzungsgrad erreicht werden, sofern die Anlage richtig eingestellt ist.[21]

2.7.3. Vorlauftemperatur, Rücklauftemperatur und Temperaturspreizung

Die Vorlauftemperatur ist die Temperatur des Heizwassers, welche nach Austritt aus dem Heizkessel gemessen wird. Die Rücklauftemperatur ist die Temperatur des abgekühlten Wassers, welches zur Heizungsanlage zurückströmt. Die Temperaturspreizung beschreibt die Differenz zwischen Vorlauf- und Rücklauftemperatur.[43]

3. Analyse

In diesem Kapitel wird auf die erste Phase des Softwareentwicklungs-Prozesses, die Phase der Analyse bzw. Definition, eingegangen. Ziel der Analyse ist es, die Anforderungen an die geplante Software gemeinsam mit dem Auftraggeber zu ermitteln und so zu beschreiben, dass Missverständnisse ausgeschlossen werden können. Diese Phase ist also ein wichtiger Teil des Projekts, da sie definiert, welches Ziel mit den darauf aufbauenden Entscheidungen und Arbeitsschritten verwirklicht werden soll.

Die Analysephase wurde in in zwei Teilschritte unterteilt. Zuerst erfolgte die Anforderungserhebung und, basierend darauf, danach die Anforderungsanalyse. Bevor diese beiden Phasen in Bezug auf die Bachelorarbeit erläutert werden, wird zunächst geklärt, was unter Anforderungen im Kontext der Softwareentwicklung zu verstehen ist, sowie welche Typen von Anforderungen es gibt.

Eine Anforderung ist eine Funktionalität oder eine Bedingung, die von einem Benutzer benötigt wird um ein Problem zu lösen oder ein Ziel zu erreichen [22, S. 62].

Des weiteren ist zu erwähnen, dass das sogenannte Requirement Engineering in der Regel nicht strikt sequentiell ist verläuft. Nachdem die Anforderungen in der Anforderungsanalyse strukturiert und formalisiert wurden, können also beispielsweise neue Anforderungen hinzukommen. Dies war auch in diesem Projekt der Fall. Die hinzugekommenen Anforderungen wurden dann ebenfalls analysiert und dem Projektziel hinzugefügt, oder im Ausblick für die zukünftige Weiterentwicklung vorgemerkt.

3.1. Anforderungserhebung

Wie bereits erwähnt, erfolgte zuerst die Anforderungserhebung. Dafür wurde in Besprechungen mit dem unternehmensinternen Zweitbetreuer evaluiert, welche Geschäftsziele von der Anwendung umgesetzt werden sollen. Diese wurden im Zuge der Anforderungserhebung notiert und inkrementell mit dem unternehmensinternen Zweitbetreuer überarbeitet. Somit konnte sichergestellt werden, dass beide Seiten das Gleiche unter den Anforderungen verstehen und diese mit den tatsächlichen Geschäftszielen und Erfordernissen übereinstimmen. An dieser Stelle soll noch einmal darauf hingewiesen werden, dass es sich bei Anforderungen um Definitionen und nicht um Tatsachen handelt.

Damit die Anforderungserhebung durchgeführt werden konnte, war es zunächst notwendig, einen Einblick in die Problemdomäne zu erhalten. Es war also ein Grund-

verständnis für Heizungsanlagen und deren Optimierungsmöglichkeiten hinsichtlich Energieeffizienz erforderlich. Dafür erfolgte zu Beginn eine Einweisung durch den unternehmensinternen Zweitbetreuer.

In der Phase der Anforderungserhebung wurden die Anforderungen unter Verwendung natürlicher Sprache in sogenannten Szenarien gesammelt.

3.1.1. Szenarien

Die im folgenden aufgeführten Szenarien stellen das Ergebnis der Anforderungserhebung dar.

- Ein Mitarbeiter der EWUS GmbH hat die Aufgabe, Maßnahmen für die Anlagenoptimierung zu planen. Um dies zu tun, müssen die vorliegenden Daten analysiert und mit den optimalen Anlagenkonfigurationen verglichen werden, um Anpassungen zu identifizieren, welche die Effizienz steigern. Da diese Maßnahmenplanung für eine große Anzahl von Anlagen regelmäßig durchzuführen ist, und außerdem teilweise sehr repetitiv und aufwendig ist, ist hier eine Automatisierung erwünscht.
- Damit die Anwendung zur Automatisierung der Ticketerstellung bereits frühzeitig genutzt werden kann, ist vorerst eine Generierung von Textbausteinen gewünscht, aus welchen schließlich manuell Tickets erstellt werden. Bevor die Textbausteine der durchgeführten Analysen in der Webanwendung Energielenker gespeichert werden, sollen Mitarbeiter die Möglichkeit haben, die Analyseergebnisse zu überprüfen und zu bearbeiten. Existieren für analysierte Anlagen bereits Textbausteine, so sollen diese neben den neu generierten angezeigt werden, sodass Mitarbeiter die neuen Textbausteine gegebenenfalls während der Überprüfung um die Alten ergänzen können.
  Das Speichern und Anpassen der generierten Textbausteine soll in einer Datenbank dokumentiert werden, damit aus einer zukünftigen Auswertung dieser Daten Informationen für die Anpassung des Analyse-Algorithmus gesammelt werden können.
- Mitarbeiter sollen die Möglichkeit haben, zu definieren, welche Analysen für welche Anlagen ausgeführt werden. Dafür soll eine Benutzeroberfläche geschaffen werden, in welcher dies auch für viele Anlagen mit möglichst geringem Zeitaufwand möglich ist.

3.1.2. Analysen

Nachdem nun das Anliegen des Auftraggebers im Allgemeinen identifiziert wurde, sollen die konkreten Analysen genauer betrachtet werden. Bei jeder der Analysen

sollen gewisse Messwerte und Kennwerte betrachtet werden und für die verschiedenen Teilmengen der Wertebereiche der betrachteten Kennwerte bestimmte Textbausteine generiert werden. Dafür wurden in Zusammenarbeit mit einem Mitarbeiter der EWUS GmbH Entscheidungsbäume für die einzelnen Analysen erstellt.
Im folgenden sollen die Analysen welche im Rahmen der Bachelorarbeit umzusetzen sind, erläutert werden. Diese werden in Zukunft um diverse weitere Analysen ergänzt. Demnach ist die Anwendung so aufzubauen, dass dies möglichst ohne große Anpassungen des entwickelten Quellcodes möglich ist.

3.1.2.1. Analyse der Anlagengröße

Diese Analyse untersucht, ob die Anlage überdimensioniert ist. Eine Anlage soll als überdimensioniert bezeichnet werden, wenn der Mittelwert ihrer Auslastung unter 80% liegt. Die Auslastung ergibt sich aus dem Anteil der gemessenen Leistung von der Nennleistung des Kessels.
Zur Berechnung der mittleren Auslastung sollen ausschließlich Messwerte von Zeitpunkten in den letzten 12 Monaten berücksichtigt werden, bei welchen die Aussentemperatur zwischen -14°C und -10°C liegt.
Ist eine Anlage überdimensioniert, so soll der folgende Textbaustein generiert werden:
„Heizkessel ist überdimensioniert, Durchschnittliche Auslastung xx%. Mögliche Maßnahmen: Brenner einstellen, andere Düse verbauen (geringere Heizleistung) oder Neubau.“

3.1.2.2. Analyse des Nutzungsgrads

Diese Analyse untersucht, ob der Nutzungsgrad der Heizungsanlagen im Sollbereich liegt. Wie bereits in Abschnitt 2.7.2 erläutert, kann bei Brennwertkesseln ein höherer Nutzungsgrad als bei Niedertemperaturkesseln erreicht werden. Demnach ist bei Definition des Sollbereichs des Nutzungsgrads eine Unterscheidung bezüglich des eingesetzten Kesseltyps notwendig.

Der Nutzungsgrad ist jeweils gemittelt über einen gewissen Zeitraum zu ermitteln. Dieser Zeitraum soll eine Mindestdauer von 14 Tagen haben und in den Monaten November bis März liegen, da in diesen die Heizung am meisten genutzt wird. Wird die Analyse also in den ersten Novemberwochen durchgeführt, so sollen die Werte des vorigen Winters betrachtet werden.

Ergibt die Analyse, dass der Nutzungsgrad nicht im Sollbereich liegt, so soll der folgende Textbaustein generiert werden:
„Die Anlage weist einen geringen Nutzungsgrad auf (Avg. Nutzungsgrad: xx%. Maßnahmen: Prüfen ob WMZ Gesamt gemessen wird, Anlagenanalyse durchführen.“

3.1.2.3. Analyse der Temperaturdifferenz

In der Analyse der Temperaturdifferenz wird die sogenannte Temperaturspreizung zwischen Vorlauf- und Rücklauftemperatur untersucht. Diese Fachbegriffe sind in Abschnitt 2.7.3 erläutert.
Die Vorlauftemperatur sollte geringstmöglich, jedoch ausreichend hoch eingestellt sein, um eine ausreichende Heizleistung zu ermöglichen. Daraus ergibt sich die Anforderung, dass analysiert werden soll, ob die mittlere Temperaturspreizung bei Brennwertkesseln mindestens 15 K und bei Niedertemperaturkesseln mindestens 10 K beträgt.
Bei dieser Analyse sollen jeweils die Werte der Temperaturdifferenz für die Monate Dezember bis Februar berücksichtigt werden. Ist die Temperaturdifferenz zu gering, so soll der folgende Textbaustein generiert werden:
„Die Temperaturspreizung ist mit xx K zu gering. Maßnahmen: Heizkurve einstellen, Absenkung VL-Temp., Verringerung der Wasserumlaufmenge."“

3.1.2.4. Analyse der Rücklauftemperatur

Anlagen, welche über einen Brennwertkessel verfügen, sollen anhand der Rücklauftemperatur hinsichtlich der ausreichenden Nutzung des Brennwerteffekts (siehe Abschnitt 2.7.2 analysiert werden.
Dafür soll untersucht werden, ob mindestens 90% der Werte der Rücklauftemperatur in den Monaten Dezember bis Februar unter 55 °C liegen. Ist dies nicht der Fall, so soll der folgende Textbaustein generiert werden: „Brennwerteffekt wird nicht ausreichend genutzt. Maßnahmen: Heizkurve einstellen, Absenkung VL-Temp., Verringerung der Wasserumlaufmenge.“

3.2. Anforderungsanalyse

Nachdem die Anforderungen zusammen mit dem Ansprechpartner im Unternehmen erhoben wurden, folgte die Anforderungsanalyse. In der Anforderungsanalyse wurden die Anforderungen entsprechend ihres Typs kategorisiert, eindeutig definiert und in atomare sogenannte User Stories überführt. Außerdem wurden die Anforderungen in dieser Phase um sogenannte nicht-funktionale Anforderungen ergänzt und auf Konsistenz und Vollständigkeit überprüft.

Eine User Story ist eine schriftliche Beschreibung einer Anforderung aus der Perspektive eines Benutzers. User Stories beinhalten jeweils die Rolle des Benutzers, welches Ziel in der User Story verfolgt wird und welchen Nutzen die User Story dem Unternehmen bringt. Sie werden in dieser Arbeit einheitlich nach folgendem Schema definiert: Als [Rolle] möchte ich [Ziel], damit [Nutzen].
Die User Stories sind bewusst in natürlicher Sprache gehalten, sodass auch die zu-

künftigen Anwender diese gut verstehen können. Im Gegensatz zu Use Cases (dt. Anwendungsfällen), sind User Stories wesentlich kompakter und weniger statisch. Nach Bedarf werden die User Stories im Zuge der agilen Software-Entwicklung genauer spezifiziert und um Akzeptanzkriterien ergänzt.[16]

Zur Verwaltung der User Stories wurde ein sogenanntes Kanban Board verwendet. Kanban ist ein Vorgehen in der agilen Software-Entwicklung, wobei Aufgaben in sogenannten Karten in Listen festgehalten werden und je nach Status in die nächste Liste verschoben werden. Diese Listen befinden sich gesammelt in einem sogenannten Kanban Board.[47]
Folgende Listen wurden dabei verwendet:

- Backlog: Hier befinden sich alle anstehenden Aufgaben bevor diese angegangen werden. Die Aufgaben werden nach Priorität sortiert angelegt.
- Todo: Hier befinden sich alle Aufgaben, welche in der aktuellen Phase angegangen werden
- Doing: Hier befinden sich alle Aufgaben, welche aktuell bearbeitet werden
- Test: Hier befinden sich alle erledigten Aufgaben, welche noch getestet werden müssen
- Review: Hier befinden sich alle Aufgaben, die umgesetzt und getestet sind, und vom Auftraggeber auf dessen Ansprüche, bzw. Erfüllung der zugehörigen Anforderungen überprüft werden müssen
- Done: Hier befinden sich die Aufgaben, die vollständig umgesetzt, getestet und abgenommen sind.

Als Software wurde dafür Trello verwendet, ein Programm in welchem Listen mit verschiebbaren Karten angelegt werden können, welche unter anderem um Checklisten für Teilaufgaben ergänzt werden können.

3.2.1. Typen von Anforderungen

Zunächst soll kurz erläutert werden, welche Typen von Anforderungen es gibt und wie diese sich unterscheiden.
Anforderungen lassen sich in funktionale und nicht-funktionale Anforderungen unterteilen. Funktionale Anforderungen beschreiben, wie sich das System verhalten soll. Sie beschreiben, wie das System auf bestimmte Eingaben und in bestimmten Situationen reagieren soll und teilweise auch, was das System nicht tun soll.
Nicht-funktionale Anforderungen hingegen beziehen sich im Gegensatz zu funktionalen Anforderungen in der Regel auf das System als Ganzes. Sie können also beispielsweise Zuverlässigkeit, Leistung und Effizienz oder Benutzbarkeit des Systems beschreiben. Sie können auch allgemeine Randbedingungen beschreiben, die von dem System zu erfüllen sind, beispielsweise die eingesetzte Programmiersprache.

In seltenen Fällen beschreiben nicht-funktionale Anforderungen auch beobachtbare Aspekte, die mit bestimmten funktionalen Anforderungen zusammenhängen.[52, S. 199ff]

3.2.2. Funktionale Anforderungen (User Stories)

Im Folgenden werden die User Stories, welche sich aus den zuvor aufgeführten Szenarios ergeben, aufgelistet. Den einzelnen User Stories werden jeweils ein Titel und, nach Bedarf, Akzeptanzkriterien zugeordnet. Die Akzeptanzkriterien sind den nicht-funktionalen Anforderungen zuzuordnen.

- **Generierung von Textbausteinen zur Anlagenoptimierung:**
  Als Ingenieur möchte ich Textbausteine für Anlagenoptimierungs-Tickets und Protokolle generieren lassen, damit ich eine manuelle Analyse und Erstellung der Textbausteine vermeiden kann.
  - Akzeptanzkriterium: Textbausteine werden ausschließlich nachdem diese akzeptiert wurden in der Webanwendung Energielenker gespeichert.
- **Anzeige voriger Textbausteine:**
  Als Ingenieur möchte ich zusätzlich zu den generierten Textbausteinen, auch eventuell bereits vorhandene Textbausteine und deren Erstellungszeitpunkt einsehen, damit ich diese miteinander vergleichen kann.
- **Speichern, Anpassen und Verwerfen von Textbausteinen:**
  Als Ingenieur möchte ich die generierten Textbausteine akzeptieren, anpassen, und verwerfen können, bevor Sie nach Energielenker geschrieben werden, damit keine falschen Tickets oder Protokolle erstellt werden.
- **Auswahl der Anlagen einzeln und gruppenweise:**
  Als Ingenieur möchte ich Anlagen für die Analyse und Konfiguration sowohl einzeln, als auch gruppenweise in der Analyse- bzw. Konfigurationsoberfläche auswählen, damit die Auswahl mit geringem Zeitaufwand geschehen kann.
- **Analyse-Konfiguration in Konfigurationsoberfläche:**
  Als Ingenieur möchte ich über eine Konfigurationsoberfläche definieren, welche Analysen für welche Anlagen ausgewertet werden, damit ausschließlich verwendbare Textbausteine erstellt werden.
- **Logging der gespeicherten Textbausteine:**
  Als Ingenieur möchte ich einsehen können, welche Textbausteine gespeichert wurden und wie diese sich von den Analyseergebnissen unterscheiden, damit diese Informationen zur Optimierung der Analyse-Konfigurationen und für zukünftige weitere Automatisierung verwendet werden können.
- **Signalisierung von durch die Datenbasis ausgelösten Fehlern:**
  Als Ingenieur möchte ich in der Analyseoberfläche auf Fehler, welche durch die

Datenbasis ausgelöst wurden, hingewiesen werden, damit diese Information bei der Entscheidung über die Interaktion mit den Textbausteinen und möglichen Anpassungen der Datenbasis oder Analyse-Konfigurationen einbezogen werden können.

3.2.3. Nicht-funktionale Anforderungen

In diesem Abschnitt werden die nicht-funktionalen Anforderungen an das System aufgeführt. Akzeptanzkriterien der zuvor aufgeführten User Stories sind ebenfalls den nicht-funktionalen Anforderungen zuzuordnen.

- **Gebrauchstauglichkeit:** Die Benutzeroberfläche soll für Ingenieure gebrauchstauglich sein und zur Verwendung keine Programmierkenntnisse erfordern.
- **Wiederverwendbarkeit:** Die Anwendung soll auf eine Art und Weise implementiert werden, dass die einzelnen Bestandteile, insbesondere die Datenbeschaffung für andere Funktionalitäten wiederverwendet werden können.
- **Erweiterbarkeit:** Es soll mit minimalen Anpassungen der Anwendung möglich sein, weitere Funktionalitäten hinzuzufügen. Insbesondere sollen neue Analysen hinzugefügt und vorhandene Analysen um weitere Bedingungen ergänzt werden können.
- **Clean Code:** Der Quellcode ist in intuitiv verständlicher Weise und entsprechend des Paradigmas „Konvention vor Konfiguration“ zu verfassen.
- **Robustheit:** Die Funktionalität des Programms soll auch bei ungünstigen Bedingungen gewährleistet sein. Insbesondere sollen von unvollständiger Datengrundlage nicht betroffene Analysen fehlerfrei ausgeführt werden.
- **Sicherheit:** Ausschließlich Mitarbeiter der EWUS GmbH sollen Zugriff auf die Anwendung haben.
- **Verwendete Technologien:** Es sollen aktuelle Technologien eingesetzt werden, welche von den Informatikern der EWUS GmbH entweder beherrscht oder leicht erlernbar sind.
- **Minimale Energielenker-Auslastung:** Die Auslastung der Datenquelle Energielenker soll minimal gehalten werden, da Mitarbeiter der EWUS GmbH häufig von Überlastungen dieser Webanwendung berichten.

4. Entwurf

In der Phase des Entwurfs wird, basierend auf den in der Analyse ermittelten Anforderungen, die zu entwickelnde Software konzipiert. Einleitend wird dafür kurz auf die Ausgangssituation und die Entscheidung über den Typ der Anwendung eingegangen. Daraufhin folgt der Entwurf der Architektur des Systems, welcher die Basis für die Qualität des Systems, also die Erfüllung der nicht-funktionalen Anforderungen bildet. Neben den Anforderungen an das System basiert der Softwareentwurf auf Entwurfsprinzipien und baut auf bewährten Lösungen, sogenannten Architekturmustern und Entwurfsmustern auf. Die angewandten Entwurfsprinzipien, Architekturmuster und Entwurfsmuster werden in Kapitel 2 genauer erklärt. In diesem Kapitel soll darauf eingegangen werden, inwiefern sie die Entwurfsentscheidungen beeinflusst haben.

Des weiteren wird in der Phase des Entwurfs auf die Auswahl der einzusetzenden Technologien und auf die Gestaltung der Benutzeroberfläche eingegangen.

4.1. Ausgangssituation

Der Entwurf des Systems geschieht „auf der Grünen Wiese “. Es gibt also große Freiheiten bezüglich der Wahl der Komponenten und Technologien. Die bisherige Infrastruktur umfasst im wesentlichen die extern eingekauften Webanwendungen Energielenker und Eneffco, deren APIs in Kombination mit einer Microsoft SQL Server Datenbank als Datengrundlage dienen. Zur Bereitstellung der Anwendungen verfügt die EWUS GmbH über virtuelle Maschinen auf einem Microsoft Server, welcher im Firmennetzwerk erreichbar ist.
Des weiteren ist hervorzuheben, dass die meisten vorhandenen und geplanten Anwendungen auf Heizungsanlagen bezogene Daten beschaffen und anpassen. Dies geschieht unter Verwendung der Datenbank und der Webanwendungen Energielenker und Eneffco.

4.2. Art der Anwendung

Als erste grundlegende Architekturentscheidung wurde über die Art der Anwendung entschieden. Wie der Titel der Bachelorarbeit bereits verrät, wurde frühzeitig entschieden, dass die Anwendung als Webanwendung umgesetzt werden soll. Da die Anwendung größtenteils am Computer verwendet werden soll, würde eine native mobile Anwendung das Ziel verfehlen. Im Gegensatz zu der Umsetzung als native Desktop-Applikation bringt die getroffene Wahl der Umsetzung als Webanwendung diverse Vorteile mit sich, welche im Folgenden kurz aufgeführt werden.

Da eine Webanwendung nicht auf dem Endgerät des Benutzers installiert werden muss, kann sie ohne zusätzlichen Aufwand auf allen Betriebssystemen und auch auf mobilen Endgerät verwendet werden und steht dem Benutzer immer in der aktuellsten Version zur Verfügung. Die Wartung der Anwendung kann also zentral auf dem Server geschehen. Ein weiterer wesentlicher Vorteil ist die große Auswahl an Frameworks und Softwarebibliotheken für die Entwicklung von Webanwendungen. [17]

Da die Anwendung ausschließlich über das Firmennetzwerk verfügbar sein soll, ist die Gefahr von unautorisierten Zugriffen gering. Auch die Verwendung der Anwendung außerhalb des Büros, beispielsweise aus dem Homeoffice ist mittels Virtual Private Network (VPN) auf sichere Weise möglich.

4.3. Architektur

In diesem entscheidenden Abschnitt des Entwurfs wird die Architektur des Systems geplant. Dabei dienen die Anforderungen als Eingabe. Insbesondere spielen die nichtfunktionalen Anforderungen, also beispielsweise die Anforderungen an die Qualität der Software eine große Rolle.
Ziel ist es, eine Architektur zu entwerfen, welche eine Erfüllung der Anforderungen ermöglicht und an den Zielen der Stakeholder orientiert ist. Wichtig ist es dabei außerdem, dass die Architekturentscheidungen mit Weitsicht getroffen werden, sodass die Software langfristig verwendet und mit minimalem Aufwand um weitere Funktionalitäten ergänzt werden kann.

Aus der Entscheidung, die Anwendung als Webanwendung umzusetzen, folgte die grundlegende Unterteilung der Architektur in ein sogenanntes Backend und Frontend in Orientierung an das Client-Server Architekturmuster (siehe Abschnitt 2.1).
Das Backend ist der Teil der Webanwendung, welcher Serverseitig läuft und für Geschäftslogik und Datenverwaltung verantwortlich ist [34]. Es stellt die Rolle des Servers im Rahmen der Client-Server-Architektur dar. Das Frontend ist der Teil der Webanwendung, welcher im Browser der Benutzer läuft und die Benutzeroberfläche bereitstellt [53]. Die Instanzen des Frontends stellen die Clients der Client-Server-Architektur dar.
Im folgenden soll nun genauer auf die Architektur des Backends eingegangen werden.

4.3.1. Architekturmuster

Eine wesentliche Entscheidung des Entwurfs der Architektur ist die Wahl des Architekturmusters anhand dessen die Anwendung entwickelt wird. Ein Architekturmuster ist eine wiederverwendbare Architektur eines Systems auf einem hohen Abstraktionslevel. Architekturmuster können für neue Anwendungen verwendet werden um Regeln und

Richtlinien für deren modulare Struktur bereitzustellen. Außerdem werden von den Architekturmustern Regeln für die Kommunikation zwischen den Modulen und deren Eigenschaften bereitgestellt. [44] Im Rahmen der Bachelorarbeit wird dabei eines der folgenden populärsten drei Architekturmuster gewählt: Monolith, serviceorientierte Architektur und Microservices. Im Kapitel 2 werden diese kurz erklärt und die serviceorientierte Architektur von dem Microservices Architekturmuster abgegrenzt. Im Folgenden wird diese Entscheidung dokumentiert und begründet.
Aufgrund der engen Kopplung der Komponenten einer monolithischen Architektur ist Wiederverwendung hier nur schwer möglich. Da Teile des zu entwickelnden Systems jedoch von diversen Anwendungen genutzt werden sollen, spricht dies stark gegen dieses Architekturmuster. Darüber hinaus ist ein Wachsen des Gesamtsystems abzusehen. Da eine Erweiterung von Monolithen nur schwer möglich ist, spricht dies ebenfalls gegen die Umsetzung als monolithische Architektur. Auch die Tatsache, dass kleine Änderungen ein neues Deployment des gesamten Systems erfordern und dies in der EWUS GmbH sehr häufig der Fall sein wird, spricht gegen die Umsetzung nach dem monolithischen Architekturmuster.
Zusammenfassend wird also das monolithische Architekturmuster als ungeeignet für das umzusetzende Projekt und dessen Perspektiven bewertet. Die Anwendung soll also als verteiltes System umgesetzt werden.

Der nächste Schritt in der Wahl des grundlegenden Architekturmusters ist die Entscheidung zwischen einer serviceorientierten Architektur und einer Microservice Architektur. Die zentrale Motivation, gewisse Funktionalitäten, insbesondere die Beschaffung der Daten der Heizungsanlagen, für verschiedene Geschäftsziele wiederzuverwenden.
Im Gegensatz zum Microservice Architekturmuster ist dies das zentrale Ziel von serviceorientierten Architekturen. Microservices lassen sich zwar ebenfalls gut wiederverwenden, sind jedoch darauf ausgerichtet, die Kopplung zu minimalisieren. Dieses Anliegen fällt aufgrund des sehr kleinen Entwicklungsteams in der EWUS GmbH weniger stark ins Gewicht.
Die hohe Komplexität die aus einer feingranularen Aufspaltung in Microservices hervorgeht, würde zu einem großen Aufwand führen, welcher nicht zwangsläufig notwendig ist. An dieser Stelle soll an das Prinzip KISS: „keep it simple stupid“ erinnert werden, welches besagt, dass keine nicht notwendige Komplexität eingeführt werden sollte, um den Entwicklungsaufwand minimal zu halten [5]. Stattdessen soll die Aufspaltung in Betracht auf die Möglichkeit, die einzelnen Services wiederzuverwenden, geschehen. Dies spricht also für die Aufteilung der Services angelehnt an die Granularität wie sie in serviceorientierter Architektur üblich ist.

Auch die Verwendung separater Datenspeicher für jeden der Microservices, wie es in der Microservice Architektur üblich ist, würde die Komplexität stark erhöhen, jedoch

für die EWUS GmbH keine Vorteile mit sich bringen. Auch dies spricht für eine serviceorientierte Architektur.

Darüber hinaus soll die Anwendung zentral auf einem Server laufen und nicht, wie es für Microservice Architekturen üblich ist, auf einer Cloud Plattform bereitgestellt werden. Die Möglichkeit, gewisse Funktionalitäten stark in Ihrer Verfügbarkeit durch mehrfaches Ausführen bestimmter Microservices zu skalieren, wird in der EUWS GmbH nicht notwendig sein. Auch hier spricht also das KISS-Prinzip für das serviceorientierte Architekturmuster.

Die Kommunikation zwischen den einzelnen Services ist bei serviceorientierten Architekturen in der Regel deutlich komplexer umgesetzt. Insbesonders kommt, wie in den Grundlagen kurz erwähnt, hier häufig eine Middleware zum Einsatz. Eine solche Middleware ist für die derzeitigen und in Zukunft anstehenden Anforderungen aber nicht notwendig und würde somit nicht notwendige Komplexität in das System einführen. Aufgrund dieser Tatsache wird für das System eine Kombination der beiden Architekturmuster gewählt. Die Anwendung wird also modularisiert wie es in serviceorientierten Architekturen üblich ist, die Kommunikation und Service Discovery jedoch umgesetzt wie in Microservice Architekturen.
An dieser Stelle ist zu erwähnen, dass die einzelnen Services zunächst unter Verwendung fester Netzwerkadressen bereitgestellt werden und keine Cloud zum Einsatz kommt. Diese Tatsache und die geringe Anzahl der Services führte zu der Entscheidung, auf einen Mechanismus zur Service Registry und Service Discovery entsprechend des Prinzips „keep it simple stupid“ (KISS) zu verzichten. Auch auf die Verwendung eines Gateways soll zunächst verzichtet werden, da keine Authentifizierung erforderlich ist und die Anzahl der Services gering ist.

Zusammenfassend wird das Backend also anhand einer Kombination der Architekturmuster Microservices und serviceorientierte Architektur umgesetzt, mit dem Ziel, Wiederverwendbarkeit zu ermöglichen und gleichzeitig die Komplexität gering zu halten. Im Wesentlichen gleicht die Architektur also einer Microservice basierten Anwendung mit einer Modularisierungsgranularität und gemeinsamen Datenspeichern, wie es in serviceorientierten Architekturen üblich ist.

An dieser Stelle soll betont werden, dass die Wahl der Architektur in Hinblick auf die Erfordernisse des sich im Digitalisierungsprozess befindlichen Ingenieurunternehmens erfolgte, und nicht in isolierter Betrachtung der im Rahmen der Bachelorarbeit umzusetzenden Funktionalitäten.

4.3.2. Teilsysteme

Nachdem das Architekturmuster gewählt wurde, ist der nächste Schritt, die konkrete Modularisierung der Anwendung zu planen. Um diesen Schritt zu dokumentieren, werden im Folgenden die einzelnen Subsysteme aufgeführt.

4.3.2.1. Facility-Service

Wie bereits erwähnt, ist das Beschaffen und Schreiben der anlagenbezogenen Daten eine zentrale Aufgabe, welche von vielen Anwendungungen wiederverwendet werden kann. Für diese Aufgabe wir ein dedizierter Service entwickelt, welcher Daten der Heizungsanlagen abfragt und zum Anpassen dieser Daten verwendet werden kann. Dieser Service wird im Folgenden, und schließlich auch in der Implementierung Facility-Service (dt. Anlagenservice) genannt.
Als Datenspeicher verwendet der Facility-Service also im wesentlichen die APIs der Webanwendungen Energielenker und Eneffco. Um die IDs der zu den einzelnen Anlagen gehörigen Energielenker-Objekte und Eneffco-Datenpunkte zu beschaffen, wird auf die zentrale Datenbank zugegriffen, welche also ebenfalls zu den Datenspeichern dieses Services gehört.
Die zentrale Aufgabe des Facility-Services ist es, JSON-Objekte für die einzelnen Anlagen zusammenzufügen, welche die in Energielenker enthaltenen Stammdaten und Parameter der Anlagen, sowie IDs der zu den Anlagen zugeordneten Datenpunkte in Eneffco beinhalten. Da die von der Anwendung zu generierenden Textbausteine zur Ticket-Erstellung ebenfalls in den Energielenker-Objekten abgespeichert werden, sind auch diese in den Anlagen-Objekten enthalten und können über eine Schnittstelle des Facility-Services angepasst werden.
Darüber hinaus kann der Facility-Service zur Abfrage der entsprechenden Eneffco-Datenpunkte verwendet werden.
Auch die Grundlage für die Anforderung, dass Benutzer eine Menge von Anlagen auswählen können, um für diese die Analysen auszuführen oder die Analyse-Konfigurationen anzupassen, wird durch eine Schnittstelle dieses Services gebildet. Diese Schnittstelle liefert eine Liste aller Anlagen-Codes zurück.

4.3.2.2. Analysis-Service

Für die Durchführung der im Zuge des Projekts umzusetzenden Analysen wird ein Service, der sogenannte Analysis-Service (dt. Analyse-Service) erstellt. Die für die Analysen benötigten anlagenbezogenen Daten erhält dieser vom Facility-Service. Auch die Textbausteine, welche als Ergebnis aus den Analysen hervorgehen werden, nachdem diese vom Benutzer akzeptiert und gegebenenfalls bearbeitet wurden, werden unter Verwendung des Anlagen-Services gespeichert. Es besteht also eine Abhängigkeit von

dem Analysis-Service zum Facility-Service. Die Kommunikation geschieht anhand der vom Facility-Service fest definierten Schnittstellen. Daher ist der Analysis-Service unabhängig von der Umsetzung des Facility-Services, verwendet jedoch dessen Funktionalität. Im Facility-Service werden die anlagenbezogenen Daten also verarbeitet und entsprechend der Vorgaben an die einzelnen Analysen ausgewertet.
Die Anforderung, dass Benutzer die Möglichkeit haben sollen, zu konfigurieren, welche Analysen für die verschiedenen Anlagen ausgeführt werden sollen, wird ebenfalls in diesem Service umgesetzt. Dafür wird dem Analysis-Service als Datenspeicher die zentrale Datenbank hinzugefügt, in welcher er diese Analysis-Konfigurationen speichert und ausliest. Somit stellt der Analysis-Service also Schnittstellen zur Konfiguration der Analysen und zum Durchführen der Analysen zur Verfügung. Diese Funktionalität wäre im Zuge eines strikt angewendeten Microservices Architekturmuster als eigener Service auszulagern. Da sie jedoch ausschließlich vom Analysis-Service verwendet wird, würde dieses Auslagern lediglich die Komplexität erhöhen, nicht aber die Wiederverwendbarkeit.

4.3.2.3. Frontend

Das Gesamtsystem besteht aus den zuvor beschriebenen Services, welche sich als Backend zusammenfassen lassen und die Rolle des Servers im Rahmen der Client-Server-Architektur umsetzen. Die graphische Benutzerschnittstelle wird im sogenannten Frontend umgesetzt, welches in den Browsern der Benutzer ausgeführt wird. Das Frontend stellt dafür Anfragen an das Backend und verarbeitet die Antworten. Die Instanzen des Frontends stellen die Clients der Client-Server-Architektur dar.

4.3.3. Kommunikationswege

Um die Kommunikation und die Abhängigkeiten zwischen den einzelnen Teilsystemen, sowie die Verwendung der Datenquellen zu veranschaulichen, wurden sogenannte UML Interaktionsdiagramme für die beiden zentralen Abläufe erstellt.

Die Unified Modeling Language (UML) ist eine graphische Standardnotation, welche zum Modellieren von Software Architekturen verwendet werden kann und dafür diverse Diagrammtypen bereitstellt. Diese lassen sich in Struktur- und Verhaltensdiagramme unterscheiden. Ein Diagrammtyp sind dabei Kommunikationsdiagramme, welche sich den Verhaltensdiagrammen zuordnen lassen.[37]
Kommunikationsdiagramme veranschaulichen den Nachrichtenfluss zwischen Objekten bzw. hier zwischen Teilsystemen und implizieren somit auch deren Beziehungen.[8]

Zum Verständnis der erstellten Kommunikationsdiagramme folgt eine kurze Erklärung der Nummerierung der Nachrichten. Die Nummerierung der Nachrichten modelliert

den zeitlichen Ablauf. Löst eine Nachricht bei ihrem Empfänger eine oder mehrere weitere Nachrichten aus, so werden diese mit Unternummern bezeichnet. Die in Abbildung 4.1 dargestellte Nachricht „3.1.1: getFacilitiesData()“ wird also von der Nachricht „3.1: analyse()“ ausgelöst. Auch die Nachricht „3.1.2: getAnalysisConfigurations()“ wird von derselben Nachricht ausgelöst. Entsprechend der letzten Unternummer wird die Nachricht mit der Nummer „3.1.2“ nach der Nachricht mit der Nummer „3.1.1“ versendet.

Die konkreten Programmierschnittstellen der Controller der einzelnen Services werden im Abschnitt 4.6.2 genauer beschrieben. An dieser Stelle soll betont werden, dass zur Vereinfachung der Darstellungen, die Namen der Nachrichten nicht mit den tatsächlichen Namen der Schnittstellen und Datenbankanfragen übereinstimmen, sondern diese lediglich als Beschreibung der Nachrichten zu interpretieren sind.

4.3.3.1. Kommunikationsdiagramm Analyse

Das in Abbildung 4.1 dargestellte Diagramm visualisiert den zentralen Ablauf, in welchem Mitarbeiter der EWUS GmbH Anlagen auswählen, für diese eine Analyse auslösen, die Ergebnisse bearbeiten und diese schließlich speichern.

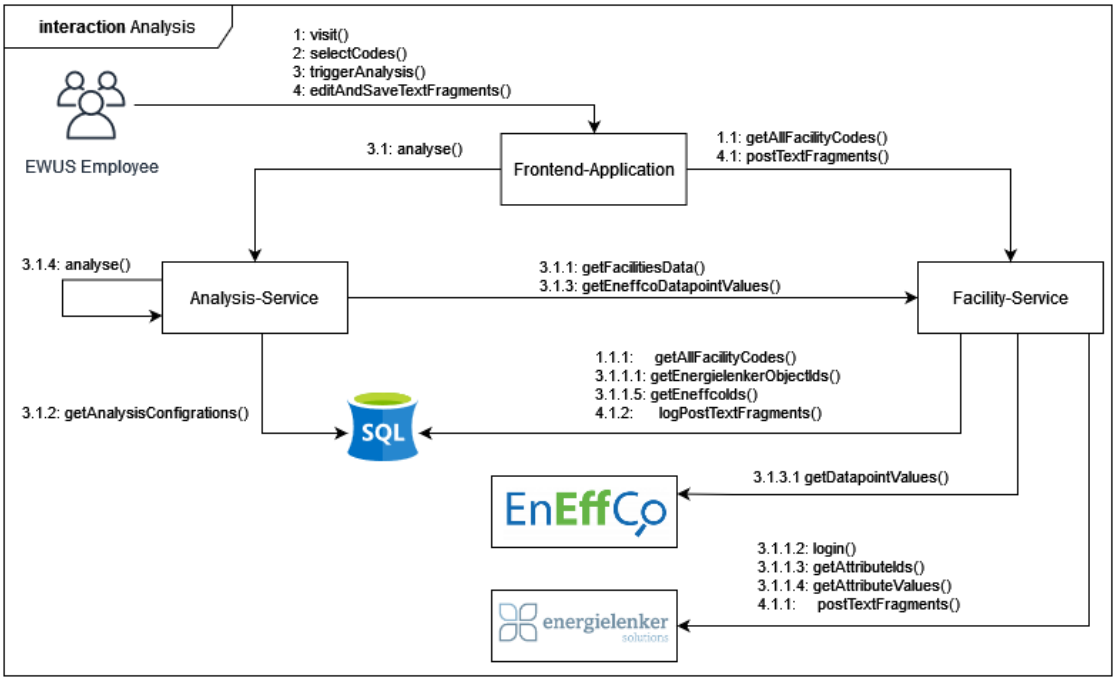

Abbildung 4.1.: *UML Kommunikationsdiagramm: Analyse; Eigene Darstellung*

4.3.3.2. Kommunikationsdiagramm Analyse-Konfiguration

Das in Abbildung 4.2 dargestellte Diagramm zeigt den Ablauf zum Anpassen der Konfigurationen ausgewählter Anlagen dar.

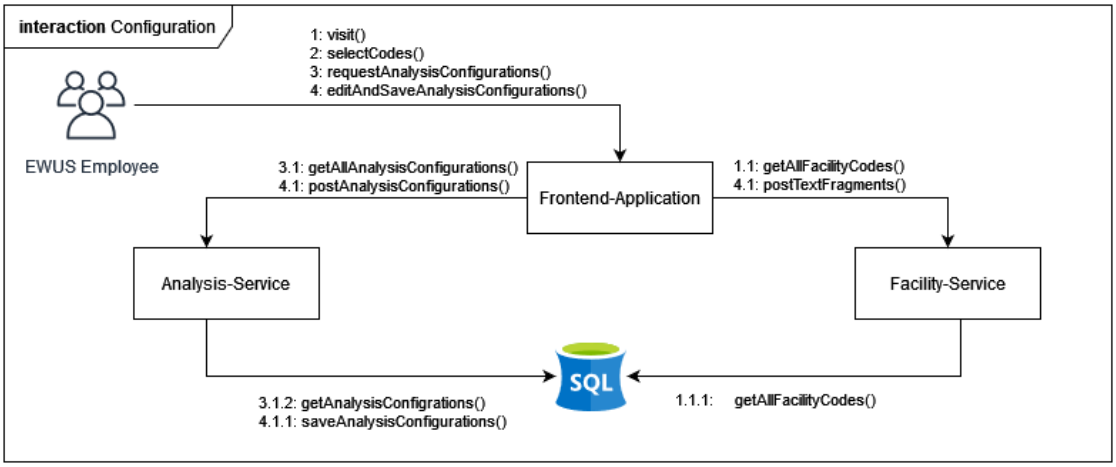

Abbildung 4.2.: *UML Kommunikationsdiagramm: Analyse-Konfiguration; Eigene Darstellung*

4.4. Technologiewahl

Nachdem auf abstrakter Ebene über den Typ der Anwendung und ihre Architektur entschieden wurde, folgte die Auswahl der zu optimalen, in der Umsetzung zu verwendenden Technologien. In diesem Abschnitt werden daher die eingesetzten Frameworks, Bibliotheken, Entwicklungswerkzeuge aufgeführt, kurz erläutert und deren Auswahl begründet.

4.4.1. Backend

Im Folgenden werden die wesentlichen Technologien, welche im Backend zum Einsatz kamen, aufgeführt.
Als Programmiersprache wird im Backend Java verwendet. Dies vereinfacht eine zukünftige Migration der bereits vorhandenen kleinen Java-Anwendungen in die Microservice-Architektur und ermöglicht den anderen Java-Entwicklern der EWUS GmbH eine einfache Einarbeitung. Wesentlich ist jedoch, dass die teilweise komplexen anstehenden Aufgaben der Firma gut mit Java gelöst und das Spring Framework mit seinen Komponenten verwendet werden kann.

4.4.1.1. Spring Framework

Um die geplante Anwendung möglichst effektiv, sicher und unter Verwendung von Features wie Dependency Injection umzusetzen wurde Spring, das am weitesten verbreitete Java Framework, eingesetzt[39].
Das Spring Framework umfasst diverse Komponenten. Die wesentlichen eingesetzten Komponenten werden im Folgenden kurz erläutert.

Spring Boot

Alle umgesetzten Services verwenden die Framework-Erweiterung Spring Boot. Spring Boot ermöglicht es gemäß des Programmierparadigmas „Konvention vor Konfiguration“ Enterprise-Anwendungen anhand eines einsatzfähigen Basisgerüstes zu entwickeln. Demnach wird sowohl Entwicklungszeit eingespart, als auch sichergestellt, dass die Anwendung für andere Entwickler gut verständlich ist und etablierte Konventionen verwendet werden [46]. Des weiteren bringt die Verwendung von Spring Boot den Vorteil mit sich, dass die Spring Boot Anwendungen eingebettete Tomcat Server beinhalten und nicht über WAR Dateien bereitgestellt werden müssen [27].

Spring Cloud

Wie in Abschnitt 4.3.1 erläutert, wird das System anhand des Microservice Architekturmusters mit einer Modularisierungs-Granularität und gemeinsamen Datenspeichern, wie es in serviceorientierten Architekturen üblich ist, umgesetzt.
Zur Verknüpfung der einzelnen mit Spring Boot umgesetzten Teilsysteme war zunächst eine Verwendung der Frameworkerweiterung Spring Cloud geplant. Insbesondere sollten Service Registrierung und Service Discovery mit einem sogenannten Spring Cloud Eureka Server umgesetzt werden. Dieser enthält Informationen über alle laufenden Service-Instanzen, welche sich als sog. Eureka-Clients bei diesem registrieren. Unter Verwendung des Eureka-Servers können diese somit ohne Angabe der URL über ihren Namen angesprochen werden.[30] In Kombination mit einem Spring Cloud Netflix Zuul Gateway kann dann schließlich über eine einzige URL auf alle Services zugegriffen werden.

Wie bereits in Abschnitt 4.3.1 erwähnt, wird jedoch zunächst, aufgrund der Bereitstellung mit festen Netzwerkadressen und der geringen Anzahl an Services, auf die von Spring Cloud unterstützten Mechanismen der Service Registry, Service Discovery und des Gateways verzichtet, um nicht notwendige Komplexität zu vermeiden.

Spring Data JPA

Spring Data JPA ist eine Spring Komponente, welche als Abstraktion einer JPA (Java Persisence API) basierten Datenschicht verwendet wird und somit eine erhebliche Einsparung von Boilerplate-Code ermöglicht[28]. Die Java Persistence API (JPA) ist ein Standard für objektrelationales Mapping (ORM), also dem Abbilden von Java-Objekten auf Tabellen und Zeilen einer Datenbank[29]. Komplexere Anfragen an die Microsoft SQL-Datenbank werden hingegen mittels Java Database Connectivity (JDBC) durchgeführt.

4.4.1.2. Testing

Die Anwendung ist mittels JUnit5 getestet. Zum Laden des Application-Contexts kommt bei Integrationstests die Annotation „@SpringBootTest“ zum Einsatz. Das Testen auf Produktivdaten sollte aus datenschutzrechtlichen Gründen vermieden werden[49]. Daher wird anstelle der Produktiv-Datenbank eine HyperSQL In-Memory-Datenbank verwendet. Produktivdaten sind Daten aus dem produktiven Einsatz einer Anwendung, und stellen das Gegenstück zu künstlich erzeugten Testdaten dar. Für das Testen der Interaktionen mit den produktiv verwendeten Webanwendungen Energielenker und Eneffco, ist aus selbigen Grund eine Simulierung dieser Interaktionen mittels Mockito geplant.

4.4.1.3. Gradle Build Tool

Als Werkzeug zur Build-Automatisierung und zum Verwalten der Abhängigkeiten kommt Gradle zum Einsatz. Alternativ ist auch eine Verwendung der Werkzeuge Maven oder Ant möglich. Gradle baut auf den Konzepten von Maven und Ant auf, und wurde aufgrund der nachfolgenden Vorteile ausgewählt.
Im Gegensatz zu Maven und Ant beinhaltet Gradle minimale Funktionalität. Benötigte Funktionalitäten wie Java Support und Spring Abhängigkeitsverwaltung werden über Plugins hinzugefügt. Daher ist Gradle in den meisten Fällen schneller als Maven. Außerdem kommt eine domänenspezifische Sprache zum Einsatz, weshalb die Konfigurationsdateien deutlich kleiner sind als die XML-basierten Konfigurationsdateien von Maven und Ant.[3]

4.4.1.4. Spotless Formatter

Um eine einheitliche Formatierung des Quellcodes zu erhalten, wurde das Gradle Plugin Spotless verwendet.

4.4.2. Frontend

Im Folgenden werden die verwendeten Technologien des Frontends aufgeführt.

4.4.2.1. React

Das Frontend wurde als Single-Page-Application mit der Open-Source JavaScript Library React entwickelt.
React ist die am weitesten verbreitetste JavaScript Bibliothek und zeichnet sich dadurch aus, dass React eine komponentenbasierte Entwicklung von Webanwendungen ermöglicht. Je nach Zustand der Webanwendung, also welche Daten vorliegen und wie die URL ist, werden verschiedene Komponenten gerendert, oder bereits dargestellte Komponenten entsprechend des akutellen Zustands aktualisiert. Die Komponenten reagieren (eng. react) also auf den aktuellen Zustand bzw. den State der Anwendung, anstelle des zeitaufwendigen Ladens vollständiger Seiten bei Benutzerinteraktionen. Daher wird auch von Single-Page-Applikationen gesprochen, da die Webanwendung aus einer einzigen Seite besteht, welche sich dynamisch anpasst.[53]

Diese Eigenschaften sind für die umgesetzte Anwendung sehr hilfreich, da alle umgesetzten Funktionalitäten auf Benutzerinteraktionen basieren. So wählt der Benutzer beispielsweise eine Menge von Anlagen aus, lässt diese analysieren. Enstprechend wird die dargestellte Tabelle erneut gerendert, sodass die Analyseergebnisse darin enthalten sind.

React Komponenten können entweder in Form von sog. Klassenkomponenten oder in der neuen Form von funktionalen Komponenten in Kombination mit React Hooks implementiert werden. Um den Empfehlungen der offiziellen Dokumentation zu folgen und möglichst zeitgemäßen Quellcode zu verfassen wurde das Frontend ausschließlich mit funktionalen Komponenten und React Hooks implementiert. Dies bringt diverse Vorteile mit, insbesondere sind die Komponenten damit leichter wiederverwendbar und in einheitlicher Syntax verfasst.[23]

4.4.2.2. Ant Design

Für das Design der Komponenten wurde die zweit-populärste React UI Bibliothek Ant Design verwendet [20]. Ant Design stellt hochwertige React Komponenten zur Verfügung. Da die Beispiele in der offiziellen Dokumentation jedoch in Form von Klassenkomponenten verfasst wurden, wurden diese als Teil der Entwicklungsarbeit als funktionale Komponenten umgeschrieben und dann entsprechend der Anforderungen konfiguriert.

4.4.2.3. ESLint Linter

Als Linter zur Identifikation von problematischen und nicht einheitlichen Quellcode-Abschnitten wurde im Frontend der erweiterbare Linter ESLint verwendet.

4.4.3. Entwicklungswerkzeuge

Es folgt eine kurze Auflistung der verwendeten Entwicklungswerkzeuge. Als Programmierumgebung wurde Visual Studio Code mit diversen Erweiterungen zur Unterstützung der verwendeten Programmiersprachen und Frameworks verwendet. Da anfänglich die Programmierung Hand in Hand mit der Einarbeitung in die verwendeten Technologien ging, wurde die Anwendung zu Beginn manuell unter Verwendung des API-Testing-Werkzeugs Postman und eines Webbrowsers getestet.

Zur Versionsverwaltung wurde ein auf Github gehostetes Git Repository verwendet. Des weiteren wurde ein Kanban-Board in Trello zur Organisation der Aufgaben verwendet.

4.5. Gestaltung der Benutzeroberfläche

Im letzten Abschnitt des Entwurfs wurde die Gestaltung der Benutzeroberfläche geplant. Dafür wurden zu Beginn die Anforderungen auf erforderliche Bedienelemente und Anwendungskontexte untersucht. Daraufhin folgte eine Benutzeranalyse, um die Benutzeroberfläche für die tatsächlichen Benutzer zu optimieren. Daraufhin wurde der Aufbau der Benutzeroberfläche per Hand skizziert, um diese dem Auftraggeber vorzustellen und gegebenenfalls anzupassen oder zu ergänzen. Schließlich wurde eine Bibliothek, welche die Bedien- und Anzeigeelemente zur Verfügung stellt, ausgewählt.

Im Zuge der Analyse der Anwendungskontexte wurde ersichtlich, dass die Anwendung in zwei verschiedenen Kontexten eingesetzt werden soll. Der zentrale Kontext ist dabei das Ausführen von Analysen für ausgewählte Anlagen. Der andere Kontext ist das Konfigurieren, welche Analysen für die einzelnen Anlagen durchgeführt werden sollen. Für das Analysieren und Konfigurieren soll jeweils eine Oberfläche erstellt werden. Da das Analysieren der deutlich frequentere Kontext ist, soll die zugehörige Oberfläche bei Aufruf der Anwendung angezeigt werden. Um zwischen den beiden Oberflächen wechseln zu können, soll der Webanwendung eine Navigationsleiste hinzugefügt werden. Entsprechend der Erwartungen der Benutzer an den Aufbau einer Webanwendungen, basierend auf deren Erfahrungen, wird die Navigationsleiste am oberen Rand der Website platziert. Da die Analyseoberfläche die Startseite der Anwendung darstellt, führt der Klick auf das, gemäß üblicher Konvention links oben in der Navigationsleiste plazierte, Logo zur Analyseoberfläche.

Die Benutzeranalyse ergab, dass die Endbenutzer der Anwendung eine kleine Gruppe von Ingenieuren der EWUS GmbH sind. Diese verfügen also über Fachwissen der Anlagenanalyse und kennen die in der Firma verwendeten Codes, welche als Bezeichner der Anlagen dienen. Des Weiteren kann davon ausgegangen werden, dass für diese Ingenieure der Umgang mit Tabellen eine sehr vertraute Tätigkeit ist. In Reaktion auf die Benutzeranalyse und die umzusetzenden Anforderungen wurden daher Tabellen als das zentrale Bedienelement der beiden Oberflächen gewählt.

Die Auswahl der zu analysierenden, bzw. konfigurierenden Anlagen geschieht unter Verwendung der Anlagencodes. Diese sind einheitlich aufgebaut nach dem Schema ABC.123 aufgebaut. Das aus drei Buchstaben bestehende Präfix definiert dabei die Firma, zu der die Anlage gehört. Die nachfolgende Zahl definiert die spezifische Anlage. Da häufig alle Anlagen einer Firma ausgewählt werden, wurde dieses Schema für die Auswahl der Anlagen genutzt. Diese wurde über eine baumartige Auswahl umgesetzt, in welcher auf oberster Ebene die Präfixes angezeigt werden und auf zweiter Ebene die einzelnen Anlagencodes. Somit können Benutzer alle Anlagen einer Firma auswählen indem sie das Präfix der Firma auswählen. Zu einer Firma gehörende spezifische Anlagen können ebenfalls schnell gefunden werden, indem das zugehörige Präfix ausgeklappt wird. Ausgewählte Anlagen sollen dem Nutzer durch Markierung der zugehörigen Elemente im Auswahlbaum, als auch durch Chips mit deren Codes im Auswahlfeld angezeigt werden.

Um die Benutzbarkeit zu maximieren, sollen die Bedienelemente einheitlich und entsprechend der Erwartungen der Benutzer gestaltet werden. Daher wird, wie bereits in der Technologiewahl erwähnt, die verbreitete UI Bibliothek Ant Design verwendet, welche hochwertige, auf optimale Gebrauchstauglichkeit ausgerichtete Bedien- und Anzeigeelemente zur Verfügung stellt. Diese haben ein in sich konsistentes Design und sind intuitiv bedienbar.
Zur Förderung der Selbstbeschreibungsfähigkeit der Software sollen für die eingesetzten Buttons Beschreibungen erscheinen, wenn sich die Maus des Benutzers auf diesen Buttons befindet.

4.5.1. Analyseoberfläche

Entsprechend des Ablaufs bei der Verwendung der Analysefunktion, wird das Element für die zu Beginn notwendige Anlagenauswahl oben positioniert. Nachdem die Analyse durch den entsprechenden Button „Analysieren“ ausgelöst wird, soll die Tabelle mit den Ergebnissen befüllt werden. Dabei soll jede Zeile die Ergebnisse einer Anlage darstellen. Da Benutzer bei der Überprüfung und Bearbeitung der generierten

Textbausteine den letzten gespeicherten Wert einsehen müssen, werden die zuletzt gespeicherten Textbausteine inklusive Zeitstempel neben den neu generierten Textbausteinen angezeigt.
Der Ladevorgang der Analyse soll dem Benutzer durch eine Ladeanimation in der Tabelle signalisiert werden.
Die Textbausteine können durch einen Bestätigungs-Button am Ende der jeweiligen Zeile, oder durch Auswahl der Zeilen gesammelt, bestätigt werden. Anschließend soll dem Benutzer eine entsprechende Erfolgs- bzw. Fehlermeldung bezüglich des Speicherns der Textbausteine in Energielenker angezeigt werden.

4.5.2. Konfigurationsoberfläche

Um minimale Komplexität in der Bedienung zu erreichen, ist der Aufbau der Konfigurationsoberfläche stark an dem der Analyseoberfläche angelehnt. Oben wird ebenfalls das Element zur Anlagenauswahl positioniert. Über einen Button „Konfigurationen Laden“ können die akutellen Konfigurationen geladen werden. Diese werden analog zu den Analyseergebnissen in einer Tabelle dargestellt. Die zu den einzelnen Anlagen gehörenden Zeilen enthalten jeweils Checkboxes für die auszuführenden Analysen. Analog zur Analyseoberfläche soll während des Ladevorgangs in der Tabelle eine Ladeanimation dargestellt werden.
Da häufig diverse Anlagen ähnlich zu konfigurieren sind, können diese gesammelt im Dialogbereich „Markierte Anlagen Konfigurieren“ konfiguriert werden. Da dies eine Aktion ist, die potenziell zu ungewollten Konfigurationen führen kann, werden die Änderungen zunächst auf die Anzeige in der Tabelle angewandt und können schließlich über den Button „Auswahl Bestätigen“ angewandt werden. Diese Funktionalität der gesammelten Konfiguration und der zweistufigen Bestätigung wurde in Absprache mit dem Auftraggeber dem initialen Entwurf der Benutzeroberfläche hinzugefügt.
Auch für das Speichern von Konfigurationen soll dem Benutzer anschließend eine entsprechende Erfolgs- bzw. Fehlermeldung angezeigt werden.

4.6. Feinentwurf Backend

Nachdem die Architektur auf abstrakter Ebene gewählt wurde, die Subsysteme identifiziert wurden, die wesentlichen zu verwendenden Technologien aufgeführt wurden und die Gestaltung der Benutzeroberfläche geplant ist, beschäftigt sich der finale Schritt des Entwurfs mit dem Feinentwurf der im Backend umzusetzenden Services. Diese wurden in Abschnitt 4.3.2 bereits identifiziert und deren wesentliche Aufgaben und Abhängigkeiten herausgearbeitet. Im Folgenden wird auf die in den Services des Backends zu verwendenden Entwurfsmuster, die umzusetzenden Schnittstellen und auf die Datenquellen eingegangen.

4.6.1. Entwurfsmuster

Ein Entwurfsmuster beschreibt ein im Softwareentwurf wiederkehrendes Problem und eine generische Lösung für dieses Problem[40]. Im Gegensatz zu Architekturmustern adressieren Entwurfsmuster die Struktur eines Subsystems und nicht die Struktur des Gesamtsystems und wie die Teilsysteme zusammenarbeiten[15].

4.6.1.1. Model-View-Controller

Als zentrales Entwurfsmuster bei der Implementierung der einzelnen Services kommt das Model-View-Controller Entwurfsmuster zum Einsatz. Der Aufbau dieses Entwurfsmusters ist im Grundlagenteil erläutert, siehe Abschnitt 2.5. In diesem Abschnitt wird darauf eingegangen, wie dieses MVC als Entwurfsmuster konkret in einzelnen Services eingesetzt werden soll.

Da die View von der Frontend Anwendung umgesetzt wird, sind die in den Services des Backends umgesetzten Funktionalitäten im wesentlichen in Controller und Models unterteilt. Die Anfragen der View werden dabei von den Controllern, welche entsprechende RESTful Schnittstellen bereitstellen empfangen und interpretiert. Diese rufen dann die benötigten Methoden der Models auf und senden schließlich eine Antwort an die View.

Die Entscheidung, das Model-View-Controller Entwurfsmuster anzuwenden, führt demnach zu der folgenden Strukturierung der Services: Im den Controllern sollen lediglich Schnittstellen bereitgestellt und die Benutzereingaben interpretiert werden. Die Geschäftslogik und Datenverwaltung soll in separaten Klassen durchgeführt werden, welche dem Model zugeordnet sind. Für die Verwaltung der Analyse-Konfigurationen wird das Model-View-Controller Entwurfsmuster um sogenannte Repositories erweitert, siehe Abschnitt 4.6.1.2.

4.6.1.2. Repository Entwurfsmuster

Das Repository Entwurfsmuster ermöglicht es, die Interaktion mit Datenquellen zu abstrahieren. Somit können verschiedene Datenquellen unterstützt werden. Der wesentliche Grund für den Einsatz dieses Patterns ist, dass das Repository Interface Frameworks ermöglicht, große Teile des Quellcodes, welcher für die Interaktion mit der verwendeten Datenquelle benötigt wird, zu generieren.

In diesem Zuge soll, wie bereits in Abschnitt 4.4 erwähnt, ein Spring Data JPA Repository als Implementierung der sogenannten Java Persistance API (JPA) zum Einsatz kommen. Dafür muss neben der Definition der Eigenschaften der Datenquelle lediglich

ein Interface definiert werden, welches JpaRepository erweitert und dabei die Datentypen der Schlüssel und der zu persistierenden Datentypen definiert.
Somit wird das zur Implementierung der Java Persistance API (JPA) benötigte Data Access Object (DAO), welches Methoden zur Interaktion mit der Datenquelle, wie findAllById, findById, save und deleteById automatisch zur Laufzeit bereitstellt, generiert.[31]

Die zur Laufzeit generierten Repositories können dann per Dependency Injection von den entsprechenden Controllern und Testklassen verwendet werden[12].

Das Repository Entwurfsmuster soll für die Verwaltung der Analysekonfigurationen zum Einsatz kommen, da diese unverändert in einer Datenbank persistiert werden sollen.

4.6.2. Programmierschnittstellen

Das Backend soll weitestgehend als RESTful API (siehe Abschnitt 2.6) umgesetzt werden.
Eine Ausnahme stellt dabei das Ausführen von Analysen dar. Diese sollen, entsprechend der Anforderungen, vom Benutzer für eine von ihm definierte Liste von Anlagen ausgelöst werden. Ein GET-Request auf die Analyseergebnisse als Ressource war dafür ursprünglich geplant, da dies der durch REST vorgeschriebenen Ressourcenorientierung entspricht. Demnach wurde also zu Beginn die Umsetztung als GET-Request an die URI "/text-fragments" geplant. Da die Ressource jedoch potenziell für eine große Menge von Anlagen ausgelöst werden soll, ist ein Request-Body notwendig, in welchem diese Anlagen spezifiziert werden. Im Zuge der Implementierung hat sich jedoch herausgestellt, dass das Spring Framework keine Request-Bodies für GET-Requests unterstützt. Daher wurde der Entwurf der entsprechenden Schnittstelle angepasst. Anstelle eines GET-Requests, welcher als Ressource die Analyseergebnisse anfragt, wurde diese als POST-Request mit der Route "/analyse" implementiert. Es handelt sich dabei also um einen sogenannten Remote Procedure Call (RPC, eng. Aufruf einer fernen Prozedur). Obwohl Remote Procedure Calls nicht den REST-Richtlinen entsprechen, wurde diese Entscheidung als legitim für den Anwendungsfall bewertet.
Als Alternative wurde dabei der Ansatz herausgearbeitet, für jede Analyse einen einzelnen REST-konformen GET-Request an die URI "/text-fragments/id" durchzuführen. Dieser Ansatz wurde aufgrund der damit verbundenen, sehr hohen Anzahl an notwendigen Anfragen an das Backend und an die Datenquellen verworfen. An dieser Stelle soll auf die nicht-funktionale Anforderung verwiesen werden, dass die Auslastung der Energielenker API minimal gehalten werden soll, da die Mitarbeiter der EWUS GmbH häufig von Überlastungen dieser Webanwendung berichten.

Die im Rahmen des Entwurfs geplanten Schnittstellen, welche als Vertrag für die Kommunikation zwischen den Teilsystemen dienen, werden im folgenden aufgeführt.

In Tabelle 4.1 werden die Schnittstellen des im Analysis-Service umgesetzten Controllers zum Verwalten von Analyse-Konfigurationen definiert.

Request	**Beschreibung**
GET /configs	Liefert alle vorhandenen Anlagen-Analyse-Konfigurationen.
GET /configs/{id}	Liefert Anlagen-Analyse-Konfiguration der spezifizierten Anlage.
GET /configs/get-list Params: codes=[“TST.001“,“TST.002“,...]	Liefert alle Anlagen-Analyse-Konfigurationen der Anlagen, welche in dem Parameter “codes“ spezifiziert sind und über eine Konfiguration verfügen.
POST /configs/{id} Body: {“id“=“TST.001“, “facilitySize“=true, “utilizationRate“=true, “deltaTemperature“=true, “returnTemperature“=true}	Speichert die im Body definierte Anlagen-Analyse-Konfigurationen.
PUT /configs/{id} Body: {“id“=“TST.001“, “facilitySize“=true, “utilizationRate“=true, “deltaTemperature“=true, “returnTemperature“=false}	Aktualisiert die in durch “id“ spezifizierte Anlagen-Analyse-Konfigurationen.
DELETE /configs/{id}	Löscht die in durch “id“ spezifizierte Anlagen-Analyse-Konfigurationen

Tabelle 4.1.: *Schnittstellen im Facility-Analysis-Configuration-Controller*

Die im Analysis-Controller des Analysis-Service umzusetzende Schnittstelle für den Remote Procedure Call, welcher die zentrale Funktionalität der Anwendung auslöst, wird in Tabelle 4.2 definiert.

Request	Beschreibung
POST /analyse Body: [“TST.001“,“TST.002“,...]	Analysiert die im Body spezifizierten Anlagen und liefert die resultierenden Textbausteine für diese zurück. Antwort-Format: {“TST.001“:[“...“,“...“],“TST.002“:...}

Tabelle 4.2.: *Schnittstellen im Analysis-Controller*

Die folgende Tabelle zeigt die Schnittstellen auf, welche in dem im Facility-Service enthaltenen Facility-Controller umgesetzt werden.

Request	Beschreibung
GET /facilities-data-list Params: codes=[“TST.001“,“TST.002“,...]	Analysiert die im Parameter “codes“ spezifizierten Anlagen und liefert die für Analysen benötigten Daten zurück. Diese umfassen in Energielenker gespeicherte Werte, sowie für die Analysen benötigte IDs von Energielenker-Objekten und Eneffco-Datenpunkten der Anlagen.
GET /facility-codes	Liefert eine Liste aller Anlagencodes.
POST /text-fragments Body: {“id“:“TST.001“, “textFragments“:“Edited“, “textFragmentsAnalysisResult“: “Original“}	Speichert die in “textFragments“ enthaltenen Textbausteine in dem Energielenker-Objekt, welches der durch “id“ spezifizierten Anlage zugeordnet ist. Erstellt zusätzlich einen Log Eintrag mit dem gespeicherten Textbausteinen und den Unterschied zu den generierten Textbausteinen.
GET /eneffco/datapoint/{id}/values Params: “from“= “2021-01-15T00:00:00.000Z“, “to“= “2021-06-30T23:59:59.000Z“, “timeInterval“=900, “includeNanValues“=false}	Liefert die Werte des spezifizierten Eneffco-Datenpunkts für den angegebenen Zeitraum. Jeder der Werte stellt einen Durchschnitt der in “timeIntervall“ definierten Dauer dar. Ist “timeIntervall“=900, so repräsentiert jeder Wert einen Zeitraum von 15 Minuten. “includeNanValues“ definiert ob die zurückgelieferte Liste gefiltert werden soll, sodass keine Nan-Werte enthalten sind.

Tabelle 4.3.: *Schnittstellen im Facility-Controller*

5. Umsetzung

In diesem Kapitel wird auf die Phase der Umsetzung, bzw. die Phase der Implementierung, des im Rahmen der Bachelorarbeit umgesetzten Prototyps eingegangen. In dieser Phase werden die ermittelten Anforderungen in Programmcode entsprechend des Entwurfs umgesetzt. Es werden also die einzelnen Teilsysteme implementiert und zum Gesamtsystem integriert. Bei der Implementierung wird ein hoher Wert auf die Einhaltung von empfohlenen Vorgehensweisen und Coding-Konventionen gelegt, sodass der Quellcode für alle Teammitglieder lesbar und bearbeitbar ist.

Zur Organisation der anstehenden und erledigten Aufgaben wurden die zugehörigen User Story Karten in die Kanban Liste, welche den jeweiligen Status signalisiert, verschoben. Teilaufgaben wurden nach Bedarf in Checklisten dieser Karten hinzugefügt und abgehakt. Auf das Kanban Vorgehen und die eingesetzten Listen wird in Abschnitt 3.2 im Zusammenhang mit User Stories eingegangen. Zusätzlich wurden einige kleine Aufgaben im Quellcode als sogenannte TODOs festgehalten.
Des Weiteren wurden zur Unterstützung einer strukturierten Umsetzung zu Beginn Methodenköpfe und Klassennamen in dem Grundgerüst der jeweiligen Anwendungen definiert, bevor diese implementiert wurden.

5.1. Clean Code

In der Umseztung wurde ein großer Stellenwert auf Clean Code gelegt.
Quellcode kann als Clean Code bezeichnet werden, wenn dieser einfach zu verstehen und ändern ist. An dieser Stelle wird auf das Buch Clean Code „Clean Code - A Handbook of Agile Software Craftsmanship“ von Robert C. Martin verwiesen, welches in diversen Hinsichten als Richtlinie zur Implementierung genutzt wurde. Zentral sind dabei die Leitsätze Konvention vor Konfiguration und keep it simple stupid (KISS).[38] Auf einige der Aspekte von Clean Code wird im folgenden in Bezug auf die Umsetzung des Prototypen eingegangen werden.

5.1.1. Konvention vor Konfiguration

Die Grundgerüste der einzelnen Teilsysteme der Anwendung wurden als erster Schritt der Implementierung vor dem Hintergrund des Clean Codes Aspekts „Konvention vor Konfiguration“ generiert.

Zur Generierung der Grundgerüste der Services des Backends, wurde die Webanwendung Spring Initializr verwendet. Zur Erstellung dieser Spring Boot Anwendungen wurden dafür jeweils die Metadaten, die wesentlichen Abhängigkeiten, sowie das zu

verwendende Build-Tool Gradle definiert.
Das Grundgerüst der Frontend Anwendung wurde mit Create-React-App initialisiert. Sowohl Spring Boot, als auch Create-React-App stellen dabei Standardkonfigurationen zur Verfügung, welche in den meisten Fällen sinnvoll sind. Dies ermöglicht eine erhebliche Reduzierung der Komplexität in der Entwicklung und erleichtert anderen Entwicklern die Anwendung zu verstehen. Somit wird durch die Anwendung des Designparadigmas „Konvention vor Konfiguration“ das KISS Prinzip (eng. keep it simple stupid) unterstützt. Entwickler, die in Zukunft an den entwickelten Services arbeiten werden, können diese, wenn sie die entsprechenden Frameworks kennen, einfach verstehen und zahlreiche Erklärungen im Internet finden, da die gängigen Konventionen eingehalten sind.

5.1.2. Quelltextformatierung

Als weiterer Clean Code Aspekt wurde auf eine einheitliche Quelltextformatierung Wert gelegt. Diese soll nicht nur innerhalb des Projekts einheitlich sein, sondern bewährten Best Practices entsprechen.
Um dies zu unterstützen, wurden Formatter und Linter verwendet und so konfiguriert, dass weit verbreitete Regeln angewandt werden. Dies bringt mehrere Vorteile mit sich. Einerseits ist somit ein automatisches Formatieren und Auflösen von problematischen Ausdrücken möglich. Andererseits werden nicht automatisch lösbare Probleme angezeigt und Entwickler können so gezwungen werden, die Regeln einzuhalten. Somit kann auch für die zukünftige Entwicklung durch andere Entwickler ein gewisses Maß an einheitlicher Quelltextformatierung gewährleiste werden.

Im Backend kam dafür, wie bereits in Abschnitt 4.4 erwähnt, das Gradle-Plugin Spotless zum Einsatz. Dieses wurde für jeden Service in der Datei „build.gradle“, welche sich auf oberster Ebene der Services befindet, eingebunden und konfiguriert. In dieser Datei werden unter Anderem die zu verwendenden Gradle-Plugins, Abhängigkeiten und Konfigurationenen der Abhängigkeiten definiert. Die Konfiguration des Spotless Gradle-Plugins umfasst im Wesentlichen die Anwendung der Richtlinien des Google Java Format Styleguides, sowie einiger Regeln bezüglich Whitespaces.

Im Frontend wurde ESLint als Linter eingesetzt, wie ebenfalls bereits in Abschnitt 4.4 erwähnt wurde. ESLint wurde konfiguriert die Richtlinien des Airbnb-Styleguides zu befolgen, und um die Plugins React, Jest, Lodash, Extra-Rules und React-Hooks erweitert. Diese Konfigurationen wurden in der Datei „.eslint.js“ vorgenommen.

5.1.3. Namensgebung

Es wurde Wert auf eine deskriptive und einheitliche Benennung von Variablen, Funktionen und Klassen gelegt. Darüber hinaus wurden sogenannte Magic Numbers (dt. Magische Zahlen), also Zahlen, deren Bedeutung schwer zu interpretieren ist, ersetzt durch entsprechend benannte Konstanten.
Methodennamen wurden im Imperativ formuliert und beschreiben, was diese tun. In Kombination mit der angestrebten Atomarität der Methoden, führt diese Namensgebung dazu, dass der Quellcode gut lesbar ist. Generell wurde angestrebt, dass der Quellcode selbstbeschreibend ist.

5.2. Kommunikation zwischen den Teilsystemen

In diesem Abschnitt wird auf die Kommunikation zwischen den Services des zu entwickelnden Systems eingegangen. Als Grundlage für die Kommunikation wurden für die Services des Backends jeweils in der Datei „src/main/resources/application.yml“ ein Port und ein Name zugeordnet. Da für die Kommunikation zunächst auf eine Service Registry verzichtet wird, werden die IP-Adresse, auf welcher die Services bereitgestellt werden und der definierte Port zur Adressierung der Services benötigt.

Die Teilsysteme kommunizieren, wie bereits erwähnt, über JSON Nachrichten, welche mittels HTTP versendet werden, miteinander. Zur Serialisierung und Deserialisierung der zu versendenden Ressourcen werden die Spring Annotation „@ResponseBody“und Instanzen der Klasse „ObjectMapper“ aus der Jackson Bibliothek verwendet. Beispiele der zu serialisierenden Objekte umfassen dabei Objekte der Klasse „Facility.java“ und der Klasse „FacilityAnalysisConfiguration.java“.
Im Folgenden werden exemplarisch zwei Codeausschnitte zur Erklärung der JSON-Basierten Kommunikation zwischen den Services und der damit verbundenen JSON-Serialisierung und -Deserialisierung aufgeführt.

Der Codeausschnitt in Listing 5.1 zeigt den Methodenkopf einer Methode, welche als Antwort eine Liste von Facility-Objekten als JSON in einer HTTP Nachricht zurückliefert. Dafür wird die Spring Annotation „@ResponseBody“ verwendet. Durch Angabe dieser Annotation wird bei Aufruf der Methode mittels HTTP Anfrage eine HTTP Antwort generiert. Diese HTTP Antwort enthält den Rückgabewert der Methode als JSON im Nachrichtenkörper. Es wird also aus der Liste von Facility-Objekten eine JSON-Array von JSON-Objekten generiert, welche jeweils die Attribute der Facility-Objekte enthalten.

```
@GetMapping("/facilities-data-list")
@ResponseBody
public ArrayList<Facility> fillFacilities(@RequestParam String codes)
```

Listing 5.1: *Methodenkopf einer Methode mit @ResponseBody Annotation*

Die soeben aufgeführte Methode des Facility-Controllers im Facility-Service wird von dem Analysis-Controller des Analysis-Service zur Beschaffung der Datengrundlage der Analysen mittels HTTP aufgerufen. Zur Deserialisierung des JSON-Arrays in eine Liste von Facility Objekten kommt die Bibliothek Jackson zum Einsatz. Die Liste wird dabei, unter Verwendung einer entsprechenden Typ-Referenz, mit einem Object-Mapper interpretiert. Diese wird exemplarisch in Listing 5.2 dargestellt.

```
ObjectMapper objectMapper = new ObjectMapper();
List<Facility> facilities = objectMapper.readValue(
    response.body().string(),
    new TypeReference(List<Facility>>(){}
);
```

Listing 5.2: *JSON Deserialisierung mittels Jackson*

5.2.1. Verwendung des Backends in der Frontend Anwendung

In der Frontend Anwendung wird zum Anfragen der benötigten HTTP-Schnittstellen das Modul Axios verwendet. Die Codes aller Anlagen, welche zu Beginn benötigt werden, werden beim Aufruf der Seite, bzw. wenn das Code-Auswahl Modul das erste Mal dargestellt wird, geladen. Dafür kommt die sogenannte useEffect Hook zum Einsatz. Dabei handelt es sich um eine Lifecycle-Methode, welche beim Laden der entsprechenden Komponente ausgeführt wird.
Weitere Anfragen an das Backend werden nach entsprechenden Benutzerinteraktionen durchgeführt. An dieser Stelle wird auf Abschnitt 4.3.3 verwiesen, in welchem die Kommunikationswege dargestellt werden.

5.3. Datenbeschaffung und -Verwaltung

In diesem Abschnitt wird auf die Datenbeschaffung und -Verwaltung eingegangen.

Die Daten, welche als Grundlage der Analyse dienen, lassen sich im Wesentlichen in zwei Kategorien unterteilen. Die erste Kategorie umfasst dabei Informationen über die Zusammensetzung der Anlagen, also beispielsweise welcher Kesseltyp verbaut ist oder an welchen Stellen Wärmemengenzähler installiert sind.
Die zweite Kategorie sind Daten, welche auf Messwerten basieren und somit jeweils

eine Menge von Zeitpunkt-Wert-Paaren darstellen. Diese Daten liegen in sogenannten Datenpunkten der Webanwendung Eneffco vor. Die Datenpunkte stellen dabei Weiterverarbeitungen und Kombinationen der verschiedenen Messwerte dar. Im Rahmen der Implementierung der Webanwendung wird dabei beispielsweise auf einen Datenpunkt, welcher den Nutzungsgrad berechnet, zugegriffen.
An dieser Stelle soll erwähnt werden, dass nach einer Besprechung der Datengrundlage mit einem Mitarbeiter der EWUS GmbH ein weiterer Eneffco-Datenpunkt angelegt wurde. In diesem wird, anhand der Vorlauf- und Rücklauftemperatur, die Temperaturspreizung berechnet und danach gefiltert, ob die Außentemperatur zwischen -14°C und -10°C liegt.
Die soeben aufgeführten Daten, welche sich jeweils auf Heizungsanlagen beziehen, werden von dem Facility-Service ausgelesen und geschrieben. Die ausgelesenen Daten, sowie die zum Schreiben der Daten notwendigen IDs werden in Objekten der Klasse Facility gespeichert und bei Anfragen an den entsprechenden Controller als JSON serialisiert zurückgeliefert. In dem Analysis-Service werden die Daten wiederum in Objekte einer Facility Klasse, welche zusätzlich über Funktionen zum Ausführen der Analyse-Geschäftslogik enthält, deserialisiert.

Zusätzlich zu der Abfrage der Daten, die in den beiden Webanwendungen vorhanden sind, werden Konfigurationen, welche Analysen für die einzelnen Anlagen ausgeführt werden sollen in einer Datenbank verwaltet. Die vom Benutzer bearbeiteten Analyse-Ergebnisse werden in der Webanwendung Energielenker gespeichert. Als letzte Komponente der Datenverwaltung ist das Logging der Bearbeitung der Analyseergebnisse zu erwähnen.
Im Folgenden wird auf die Besonderheiten bei der Datenbeschaffung und -verwaltung eingegangen.

5.3.1. Energielenker

Die Daten, welche in Energielenker vorliegen, sind in einer baumartigen Struktur gespeichert. Die Knoten dieses Baumes werden im Folgenden Energielenker-Objekte genannt. Für jede Anlage gibt es als Wurzelelement ein sogenanntes Liegenschafts-Objekt. Dieses enthält unter anderem den Anlagencode, den Versorgungstyp und einen Code, der die zugeordnete Außentemperatur repräsentiert.
Des weiteren wird im Rahmen der umzusetzenden Anwendung auf jeweils ein Kind dieser Liegenschafts-Objekte zugegriffen, das Einsparzählerprotokoll-Objekt. Die vom Benutzer bearbeiteten, generierten Textbausteine werden schließlich in einem Kind dieses Einsparzählerprotokoll-Objekts, dem Regelparameter_Soll-Werte-Objekt gespeichert.
Die Baumstruktur der Energielenker-Objekte der Anlagen ist in Abbildung 5.1, einem

Bildschirmfoto der Energielenker Webanwendung ersichtlich.

Abbildung 5.1.: *Baumstruktur der Energielenker-Objekte; Bildquelle: [14]*

5.3.1.1. Ermittlung der Energielenker-Objekt-IDs

Damit unter Verwendung der Energielenker API auf die Energielenker-Objekte zugegriffen werden kann, werden deren IDs benötigt. Diese können unter Verwendung der zentralen Datenbank abgefragt werden. Dafür kann die Tabelle „energielenker_sortiert“ verwendet werden. In dieser Tabelle sind die IDs der Energielenker-Objekte, deren Namen und die zugehörigen Anlagencodes aufgeführt. Über die Spalte des Namens lässt sich ermitteln, ob es sich um Einsparzählerprotokoll-Objekte oder Regelparameter_Soll-Werte-Objekte handelt. Eine Herausforderung ist jedoch, dass bei Liegenschaftsobjekten in dieser Spalte lediglich die Adresse der jeweiligen Liegenschaft gespeichert ist, nicht aber der Typ des Energielenker-Objekts. Darüber hinaus sollen lediglich Energielenker-Objekte von Anlagen, welche nicht als deinstalliert markiert sind, abgefragt werden. Um diese Herausforderungen zu lösen, wurde im Rahmen der Bachelorarbeit der in Abbildung 5.3 dargestellte SQL-Befehl entwickelt.

```
SELECT
    [energielenker_sortiert].[code],
    [energielenker_objects].[name],
    [energielenker_sortiert].[id],
    [energielenker_objects].[parentId]
FROM
    [energielenker_sortiert] JOIN [energielenker_objects]
        ON [energielenker_sortiert].[id] = [energielenker_objects].[id]
WHERE
    [code] IN ('ABC.001', 'ABC.002', 'ABC.003')
    AND [deinstalled] = 'false'
    AND [energielenker_objects].[name]
        IN ('Einsparzählerprotokoll', 'Regelparameter_Soll-Werte');
```

Listing 5.3: *SQL-Befehl zum Abfragen und Zuordnen der Energielenker-Objekte*

Im wesentlichen wird also ein Inner Join (dt. innerer Verbund) mit der Tabelle „energielenker_objects“ mit der Verbundbedingung, dass die IDs übereinstimmen durchgeführt. In der Tabelle „energielenker_objects“ wird in der Spalte „parentId“ das Elternelement des jeweiligen Energielenker-Objekts referenziert. Da, wie in Abbildung 5.1 ersichtlich ist, die Einsparzähler-Objekte direktes Kind des jeweils zugehörigen Liegenschafts-Objekts sind, können die IDs der Liegenschafts-Objekte also anhand der Einsparzählerprotokoll-Objekte „parentId“ ermittelt werden.
Auch das Filtern nach Anlagen, welche nicht deinstalliert sind, ist unter Verwendung dieser Tabelle möglich, da diese Information der Spalte „deinstalled“ entnommen werden kann.

Der in Abbildung 4.2 aufgeführte SQL-Befehl wird in der Methode „fillEnergielenkerObjectIds(Connection dbConnection, ArrayList<Facility> facilities)“ ausgeführt. Anstelle der zur Illustration eingetragenen Anlagencodes werden dynamisch die Anlagencodes der zu befüllenden Facility-Objekte verwendet.

5.3.1.2. Verwendung der Energielenker API

Damit, unter Verwendung der Energielenker API, Werte der Energielenker-Objekte gelesen und geschrieben werden können, werden zusätzlich zu den Energielenker-Objekt-IDs auch IDs der entsprechenden Attribute verwendet. Um diese Attribut-IDs zu erhalten, wird für die jeweiligen Energielenker-Objekte eine Schnittstelle der Energielenker API verwendet, welche Informationen über die Attribute eines Energielenker-Objekts zurückliefert. Die Attribut-IDs können schließlich durch Filtern der Antwort anhand der Namen der benötigten Attribute ermittelt werden. Die URL dieser REST-Schnittstelle ist „https://ewus.elmonitor.de/api/v1/basemonitor/objects/{obj-ID}/attributes“.

Die Werte der entsprechenden Attribute werden über die Schnittstelle „https://ewus.elmonitor.de/api/v1/basemonitor/objects/{obj-ID}/attributes/{attr-ID}“ abgefragt und über dieselbe URI aktualisiert.

Abschließend ist zu erwähnen, dass zur Verwendung der Energielenker API eine Anmeldung notwendig ist. Dafür werden die Benutzerdaten unter Verwendung der Klasse „Config.java“ von einer Datei, welche die Anmeldedaten enthält, ausgelesen und damit ein Anmelde-Request durchgeführt. Diese Datei ist in der „.gitignore“ aufgelistet, sodass sie nicht auf den Github-Server hochgeladen wird.
In der Antwort wird ein JSON Web Token zurückgeliefert, welches in nachfolgenden Anfragen zur Authentifizierung im Authorization-Header mitgesendet wird.

5.3.2. Eneffco

Auch zur Abfrage der Datenpunkte von Eneffco werden für jede Anlage die entsprechenden Datenpunkt-IDs benötigt. Damit diese ermittelt werden können, müssen zu Beginn für die einzelnen Anlagen die Codes dieser Datenpunkte bestimmt werden. Diese ergeben sich aus dem Anlagencode, einem Code, welcher die Art des Datenpunkts spezifiziert und der Nummer des Wärmemengenzählers, welche zuvor aus Energielenker ausgelesen wird. Daraus ergibt sich beispielsweise für den Temperaturdifferenz-Datenpunkt der Anlage mit dem Code ABC.001, welche über genau einen Wärmemengenzähler verfügt der Datenpunkt-Code „ABC.001.WEZ.WMZ.DT.1“.

Anhand dieses Datenpunkt-Codes kann schließlich aus der Datenbank-Tabelle „eneffco_datapoint“ die ID dieses Datenpunkts abgefragt werden.

5.3.2.1. Verwendung der Eneffco API

Sind die benötigten Datenpunkt-IDs vorhanden, so ist zunächst eine Authentifizierung notwendig. Da die für die Eneffco API zur Authentifizierung benötigten Tokens keine Begrenzung in der Zeit Ihrer Gültigkeit haben, wird ein entsprechendes Token aus der Konfigurationsdatei ausgelesen. Dieses wurde einmalig unter Angabe von Benutzername und Passwort mit einem Base64-Kodierer generiert und analog zu den Energielenker Anmeldedaten unter Verwendung der Klasse „Config.java“ aus einer nicht mit Git versionierten Konfigurationsdatei ausgelesen.

Wie bereits in der Kategorisierung der zur Analyse notwendigen Daten erwähnt, handelt es sich bei den aus Eneffco ausgelesen Daten um Zeitpunkt-Wert-Paare. Entsprechend ist für die Abfrage der Datenpunkte ein Zeitraum, sowie ein Zeitintervall notwendig. Der durch einen Start- und Endzeitpunkt zu definierende Zeitraum wird im Rahmen der jeweiligen Analysen auf verschiedene Weisen berechnet. Wesentliche

Faktoren sind dabei die Bestimmung der Monaten die Datenpunkte Werte enthalten, welche zur Analyse geeignet sind, sowie eine Mindestdauer des zu überprüfenden Zeitraums. Die Zeitintervalle dienen der Diskretisierung der nahezu kontinuierlich vorliegenden Werte. Jeder zurückgelieferte Wert entspricht also dem Mittelwert einer durch das angegebene Zeitintervall definierten Zeitspanne.
Sowohl die für die jeweiligen Analysen zu verwendenden Zeitintervalle, als auch die Logik zur Bestimmung der zu zu verwendenden Zeiträume wurden in Absprache mit einem Mitarbeiter der EWUS GmbH definiert.

5.3.2.2. Interpretation der Werte-Mengen

Je nach Analyse sind die von der Eneffco API zurückgelieferten Werte-Mengen der entsprechenden Datenpunkte verschieden zu verarbeiten. Für die Analyse des Nutzungsgrads, der Anlagengröße und der Temperaturdifferenz wird das arithmetische Mittel dieser Werte gebildet. Für die Analyse der Rücklauftemperatur wird der Anteil der Werte bestimmt, welcher in einem akzeptablen Bereich liegt.

Die bestimmten arithmetischen Mittel bzw. der Anteil der akzeptablen Werte werden schließlich anhand von zuvor ermittelten Kriterien untersucht. Diese Kriterien wurden im Rahmen des zu entwickelnden Prototyps fest definiert. Da die Kriterien jedoch basierend auf der Qualität der Analyseergebnisse zu optimieren sind, sollen diese schließlich aus der Datenbank ausgelesen werden, sodass sie ohne Programmierkenntnisse von den Ingenieuren der EWUS GmbH selbständig angepasst werden können.

5.3.3. Analyse-Konfigurationen und Logs

Wie bereits im Entwurf erwähnt (siehe Abschnitt 4.6.1.2), wird zur Verwaltung der Analyse-Konfigurationen ein Spring Data JPA Repository verwendet. Als erster Schritt wurde dafür die von dem Repository zu verwendende Datenquelle in der Datei „src/-main/resources/application.yml“ definiert.
Des weiteren wurde ein Interface definiert, welches das generische JpaRepository erweitert und dabei als Datentyp der zu persistierenden Daten die Klasse FacilityAnalysisConfiguration und als Datentyp der Schlüssel die Klasse String definiert. Die zu persistierende Klasse FacilityAnalysisConfiguration wird durch eine Annotation als sogenannte Entity markiert.
Die zu persistierenden Attribute dieser Klasse werden ebenfalls mit einer Annotationen versehen, über welche definiert wird, welche Spalten verwendet werden, und bei welchem Attribut es sich um den Schlüssel handelt.

Das, basierend auf den benötigten Methoden, zur Laufzeit generierte Repository wird

schließlich per Dependency Injection in dem Controller zur Verwaltung der Analyse-Konfigurationen eingebunden.

Auch für das Speichern von Logs, welche die vom Benutzer bearbeiteten Textbausteine, die zugehörigen unbearbeiteten Analyseergebnisse, den Unterschied zwischen diesen Textbausteinen, sowie zugehörigen Anlagencode und Zeitstempel enthalten, wird dieselbe Datenbank verwendet. Da diese Logs jedoch lediglich geschrieben werden, kommen dabei zur Reduktion der Komplexität lediglich entsprechende SQL-Statements zum Einsatz.

5.4. Testing

Um die korrekte Arbeitsweise der Anwendung zu gewährleisten, wurden im Rahmen der Bachelorarbeit automatisierte Tests entwickelt. Diese lassen sich im wesentlichen in Unit Tests und Integration Tests unterteilen, auf welche im Folgenden separat eingegangen wird.
Wie bereits in Abschnitt 4.4.1.2 erläutert kamen dabei JUnit5, Mockito und eine HyperSQL In-Memory-Datenbank zum Einsatz.
Die Einarbeitung in diese Technologien erfolgte jedoch erst zu einem späteren Zeitpunkt der Implementierung. Aus diesem Grund, und, da zunächst ein Verständnis über die zu verwendenden APIs und über die Datenbankstruktur erarbeitet werden musste, wurden zu Beginn der Implementierung manuelle Tests durchgeführt. Dafür kamen Konsolen-Ausgaben in Kombination mit dem API-Testwerkzeug Postman und der Frontend Anwendung. Um den Aufwand des manuellen Testens zu minimieren wurde in Postman eine Bibliothek der benötigten Anfragen angelegt.

5.4.1. Unit Tests

In Unit Tests werden kleine Einheiten des Quellcodes getestet. Sie dienen der Verifizierung der korrekten Umsetzung von einzelnen Methoden und Modulen. Da in diesen Tests lediglich die Funktionsweise dieser Einheiten getestet werden soll, werden andere Abhängigkeiten simuliert. Datenbankzugriffe und Aufrufe von Methoden oder Schnittstellen außerhalb dieser Einheiten werden daher in den Unit Tests mit sogenannten Mocks simuliert. Dabei wird getestet, ob die notwendigen Aufrufe der simulierten Abhängigkeiten durchgeführt werden. Benötigte Antworten auf diese Aufrufe werden simuliert.[19]

5.4.2. Integrationstests

Bei Integrationstests werden im Gegensatz zu Unit Tests mehrere Module der Anwendung kombiniert und als Gruppe getestet. Dabei wird die Schnittstellen und die

Interaktion zwischen den Modulen verifiziert.[19]
Zur Durchführung der Integrationstests wird unter Verwendung der Annotation „@SpringBootTest“ der Kontext der mit Spring Boot umgesetzten Services geladen.

6. Demonstration und Evaluierung

In diesem Kapitel werden zu Beginn die Ergebnisse der Bachelorarbeit anhand der Benutzeroberfläche demonstriert und dabei die Umsetzung der funktionalen Anforderungen evaluiert. Anschließend werden die nicht-funktionalen Anforderungen evaluiert. Nachdem die Ergebnisse demonstriert und bezüglich der Anforderungen evaluiert wurden, wird das Vorgehen evaluiert.

6.1. Demonstration und Evaluierung der funktionalen Anforderungen

Es folgt eine Demonstration des im Rahmen der Bachelorarbeit entwickelten Prototyps zur automatisierten Ermittlung von Optimierungsmöglichkeiten in Heizungsanlagen. Im Zuge der Demonstration der Benutzeroberfläche wird dabei die Erfüllung der funktionalen Anforderungen evaluiert.

6.1.1. Auswahl der Anlagen

Die Auswahl der zu analysierenden, bzw. zu konfigurierenden Anlagen erfolgt jeweils zu Beginn der Abläufe. Die entsprechende Komponente, welche sowohl in der Analyseoberfläche, als auch in der Konfigurationsoberfläche zum Einsatz kommt, wird in Abbildung 6.1 dargestellt. Die Anlagen können dabei aus einer baumartigen Auflistung der Anlagen entweder einzeln oder gruppenweise durch Auswahl eines Präfixes ausgewählt werden. Diese Präfixe der Anlagencodes repräsentieren dabei alle Anlagen des durch das jeweilige Präfix definierten Kunden. Die Anforderung „Auswahl der Anlagen einzeln und gruppenweise“ ist damit erfüllt.

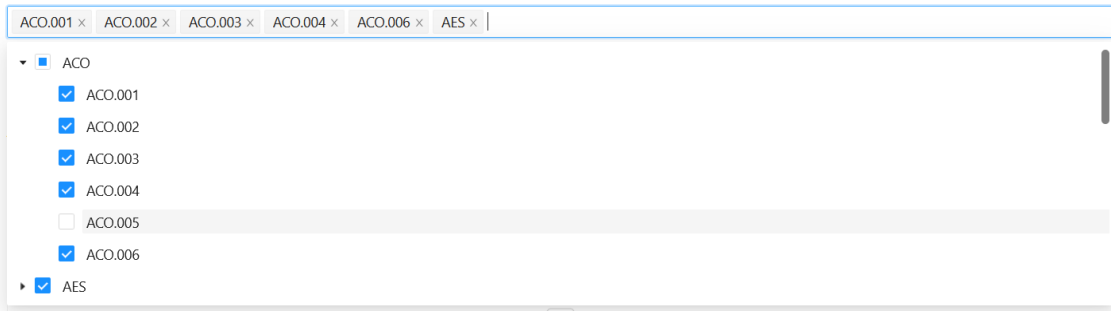

Abbildung 6.1.: *Demonstration: Auswahl der Anlagen; Bildschirmfoto (eigene Darstellung)*

6.1.2. Analyseoberfläche

In Abbildung 6.2 ist die zentrale Oberfläche der Webanwendung, welche zur Generierung der Textbausteine zur Anlagenoptimierung genutzt werden kann, dargestellt. Die in der Komponente zur Auswahl der Anlagen ausgewählten Anlagen können durch Klicken des entsprechenden Buttons analysiert werden. Anschließend wird die Tabelle, welche die Ergebnisse der Analyse darstellt mit den entsprechenden Textbausteinen befüllt. Jede Reihe stellt dabei die für eine Anlage generierten Textbausteine, sowie vorige Textbausteine mit deren Erstellungszeitpunkt dar. Die Textbausteine können bearbeitet werden und schließlich entweder einzeln oder über den Button „Auswahl bestätigen“ in Energielenker gespeichert werden.
Damit sind die Anforderungen „Generierung von Textbausteinen zur Anlagenoptimierung“, „Anzeige voriger Textbausteine“ erfüllt. Auch die Anforderung „Speichern, Anpassen und Verwerfen von Textbausteinen“ ist erfüllt, da Textbausteine, welche nicht gespeichert werden, verworfen werden.

Aufgrund der erforderlichen Bestätigung zum Speichern der Textbausteine durch die Buttons „Bestätigen“, bzw. „Auswahl bestätigen“, ist auch das Akzeptanzkriterium, dass Textbausteine ausschließlich nachdem diese akzeptiert wurden, in Energielenker gespeichert werden, erfüllt.
Bestätigt der Benutzer die Textbausteine einer Anlage, so werden diese in einem Feld eines Energielenker-Objekts dieser Anlage gespeichert. Darüber hinaus wird beim Speichern von Textbausteinen ein Log-Eintrag in der zentralen Datenbank erstellt. Dieses Log enthält jeweils die gespeicherten, möglicherweise bearbeiteten Textbausteine, die generierten Textbausteine, den Unterschied zwischen diesen, sowie einen Zeitstempel und den Code der Anlage. Somit ist auch die Anforderung „Logging der gespeicherten Textbausteine“ erfüllt.

Treten bei den Analysen Fehler auf, welche durch fehlende oder nicht auswertbare Daten ausgelöst werden, so werden entsprechende Textbausteine generiert und

dem Benutzer angezeigt. In diesen Textbausteinen wird der Benutzer aufgefordert, die Datenbasis zu überprüfen. Die Anforderung „Signalisierung von durch die Datenbasis ausgelösten Fehlern“ ist damit ebenfalls erfüllt.

Zuletzt ist bezüglich der Analyseoberfläche zu erwähnen, dass bei erfolgreichem Speichern eine entsprechende Erfolgsmeldung und anderenfalls eine Fehlermeldung angezeigt wird.

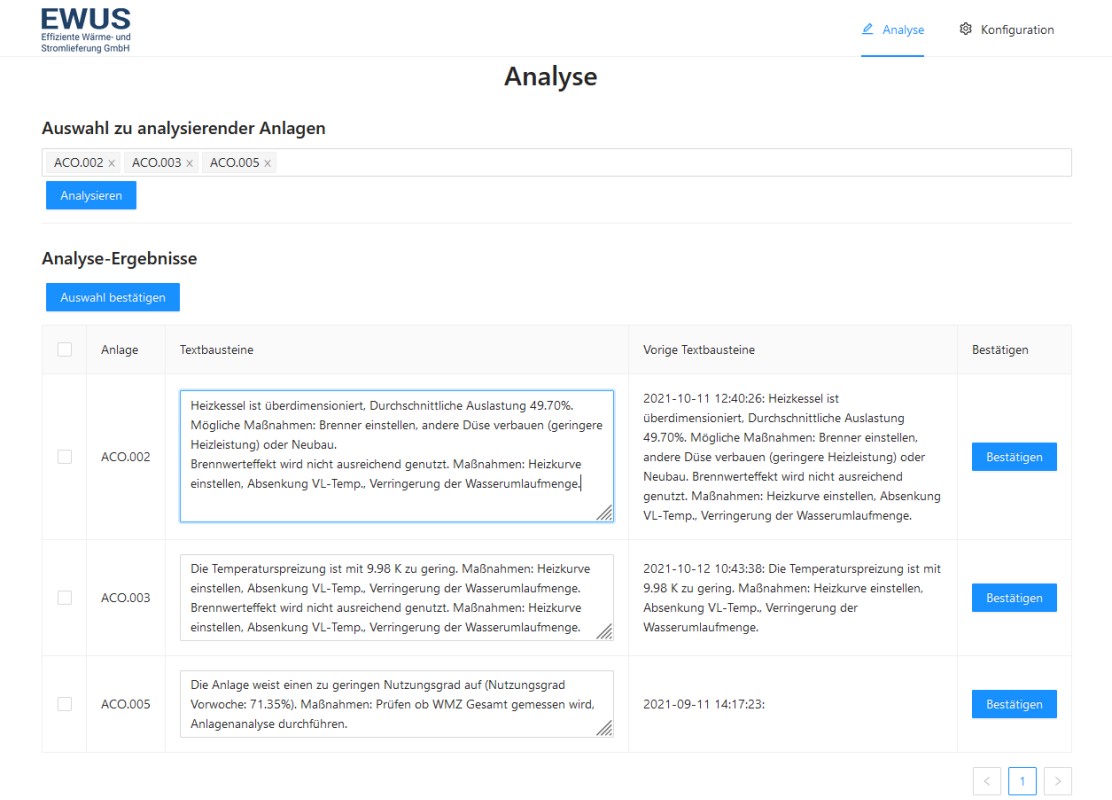

Abbildung 6.2.: *Demonstration: Analyseoberfläche; Bildschirmfoto (eigene Darstellung)*

6.1.3. Konfigurationsoberfläche

Über das Menü in der Kopfleiste der Webanwendung kann zu der Konfigurationsoberfläche navigiert werden. Diese ist in Abbildung 6.3 dargestellt. Dort können für die vom Benutzer ausgewählten Anlagen Konfigurationen geladen und angepasst werden. Der Aufbau dieser Oberfläche ist stark an den der Analyseoberfläche angelehnt. Anstelle von Textbausteinen werden in der Tabelle dabei die einzelnen Analysen mit zugeordneten Checkboxes angezeigt, über welche diese aktiviert bzw. deaktiviert werden können. Somit ist die Anforderung „Analyse Konfiguration in Konfigurationsoberfläche“ erfüllt. Auch in dieser Oberfläche müssen Änderungen einzeln oder gruppenweise bestätigt werden. Des weiteren ist zu erwähnen, dass die Analysen auch für mehrere Anlagen

gleichzeitig angepasst und anschließend bestätigt werden können. Dies ist über die Option „Markierte Anlagen Konfigurieren“ möglich.

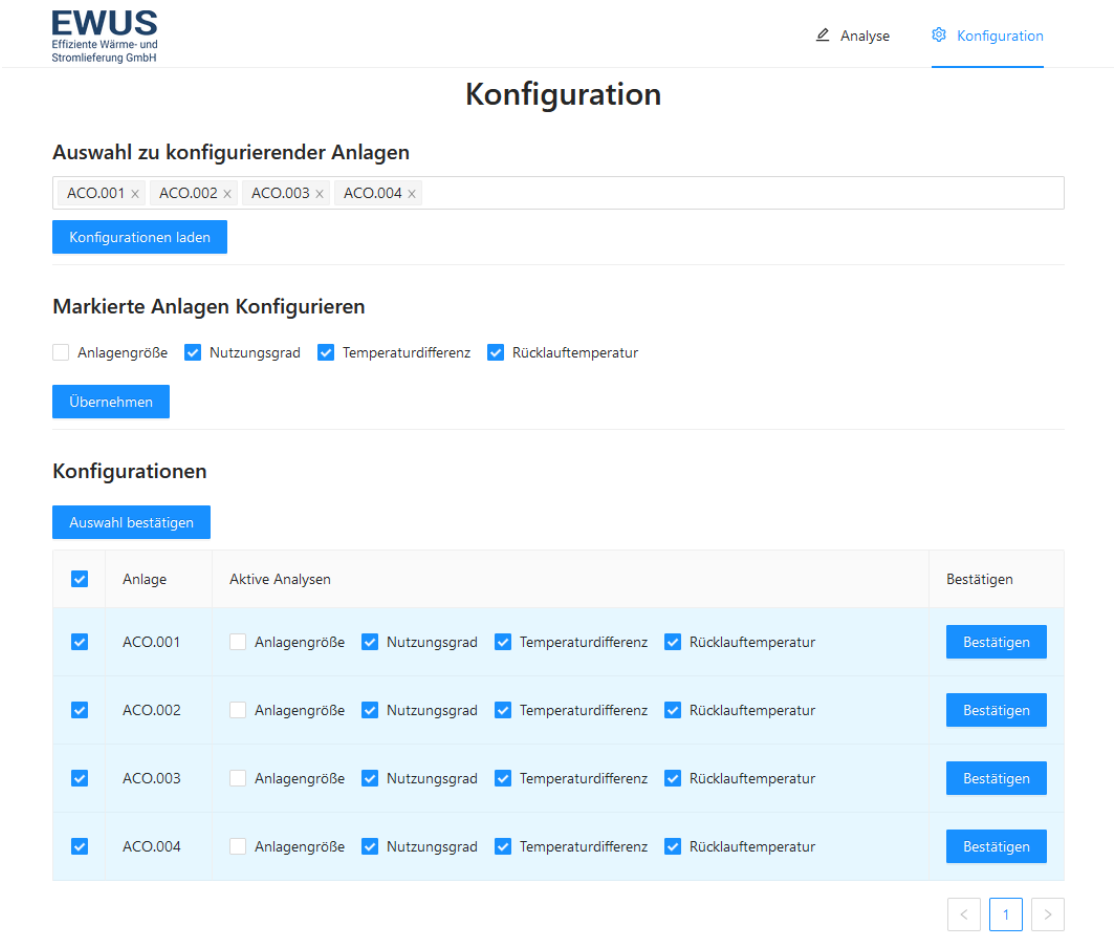

Abbildung 6.3.: *Demonstration: Konfigurationsoberfläche; Bildschirmfoto (eigene Darstellung)*

Wird eine Große Anzahl an Anlagen ausgewählt, so kann der Benutzer anpassen, wie viele Konfigurationen pro Seite angezeigt werden, siehe Abbildung 6.4. Ebenso ist in der Analyseoberfläche diese Konfigurierung der Paginierung möglich.

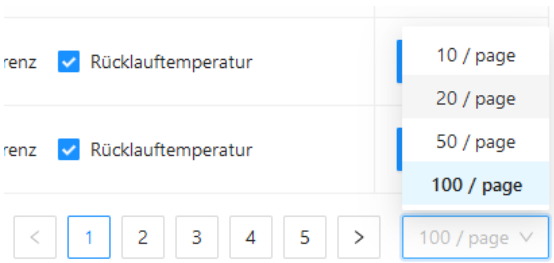

Abbildung 6.4.: *Demonstration: Konfigurierbare Paginierung; Bildschirmfoto (eigene Darstellung)*

6.2. Evaluierung der nicht-funktionalen Anforderungen

Nachdem im vorigen Abschnitt die funktionalen Anforderungen evaluiert wurden, folgt in diesem Abschnitt eine kurze Evaluierung der nicht-funktionalen Anforderungen.

6.2.1. Wiederverwendbarkeit

Das umzusetzende System wurde im Wesentlichen als Service orientierte Architektur umgesetzt, siehe Abschnitt 4.3.1. Somit ist unter anderem der sogenannte Facility-Service, in welchem die Datenbeschaffung erfolgt auch für andere Funktionalitäten wiederverwendbar. Auch die Modularisierung innerhalb der Teilsysteme fördert die Wiederverwendbarkeit. Somit kann die Anforderung der „Wiederverwendbarkeit“ als erfüllt betrachtet werden.

6.2.2. Erweiterbarkeit

Die Anforderung der „Erweiterbarkeit um weitere Analysen“ wird ebenfalls als erfüllt betrachtet. Gründe für diese Bewertung umfassen die Wahl des SOA Architekturmusters, die Modularisierung der Teilsysteme und die Orientierung an dem Prinzip keep it simple stupid (KISS). Auch das Hinzufügen und Erweitern von Analysen ist mit geringen Aufwand möglich.

6.2.3. Clean Code

Bezüglich der Erfüllung der Anforderung „Clean Code“ wird auf Abschnitt 5.1 verwiesen. Zusammenfassend lässt sich diesbezüglich hervorheben, dass das Prinzip „Konvention vor Konfiguration“ berücksichtigt wurde, Werkzeuge zur Unterstützung einheitlicher Quelltextformatierung zum Einsatz kamen, und die Namensgebung und Modularisierung entsprechend gängiger Konventionen erfolgten. Demnach ist diese Anforderung im Wesentlichen als erfüllt zu betrachten.

6.2.4. Sicherheit

Die Anforderung, dass ausschließlich Mitarbeiter der EWUS GmbH Zugriff auf die Anwendung haben, wird damit erfüllt, dass die Webanwendung auf einem Server bereitgestellt wird, welcher ausschließlich aus dem Firmennetzwerk erreichbar ist. Auf ein Authentifizierungsverfahren wurde auf Anweisung des Auftraggebers vor diesem Hintergrund verzichtet.

6.2.5. Robustheit

Die Anforderung der Robustheit ist nur teilweise als erfüllt zu bewerten. Der Teil der Anforderung, dass bei unvollständiger Datengrundlage nicht betroffene Analysen fehlerfrei ausgeführt werden, ist erfüllt. Ist die benötigte Datengrundlage nicht vollständig, so wird dem Benutzer dies durch einen entsprechenden Textbaustein signalisiert.
Für die vollständige Erfüllung der Anforderung der „Robustheit“ ist ein umfassenderes Abfangen von Sonderfällen, sowie ausgiebigeres Testing erforderlich. An dieser Stelle soll betont werden, dass es sich bei dem entwickelten System um einen Prototyp handelt, welcher unter anderem dafür genutzt werden soll, abzufangende Randbedingungen und Analysekriterien zu ermitteln. Insbesondere die Frontend Anwendung ist eine Übergangslösung, da das angestrebte Endprodukt, anstelle von manuell zu überprüfenden Textbausteinen, vollautomatisiert Tickets zur Anlagenoptimierung generieren soll. Aus diesem Grund wurde das Frontend lediglich ausgiebig manuell getestet.

6.2.6. Verwendete Technologien

Diese Anforderung wird als umgesetzt bewertet, da die eingesetzten Frameworks aktuell sind und die Entwickler der EWUS GmbH bereits die Programmiersprache Java beherrschen. Zum Verständnis der Anwendung muss also lediglich eine Einarbeitung in das Spring Framework erfolgen, was anhand des entwickelten Prototyps mit geringem Aufwand möglich ist. An dieser Stelle wird auf Abschnitt 4.4 verwiesen, in welchem die Auswahl der zentralen Technologien begründet wird.

6.2.7. Minimale Energielenker-Auslastung

Auch die Anforderung, dass die Auslastung der Datenquelle Energielenker minimal gehalten werden soll, wird als erfüllt betrachtet. Dabei ist zu erwähnen, dass Daten, welche sowohl in der Datenbank, als auch über die Energielenker API verfügbar sind, aus der Datenbank bezogen werden. Die Anfragen an die Energielenker API sind auf ein Minimum reduziert.

6.2.8. Gebrauchstauglichkeit

Er Evaluierung der Anforderung an die Gebrauchstauglichkeit wurde die Anwendung den zukünftigen Benutzern ohne weitere Erklärungen zum Testen vorgelegt. Anschließend wurden diese gebeten, die Kurzversion des IsoMetrics Fragebogens zur Evaluation von graphischen Benutzungsschnittstellen[56] auszufüllen. Da lediglich zwei Mitarbeiter die Benutzungsschnittstelle verwenden werden, ist eine empirische Auswertung des Fragebogens schwer möglich. Daher wurde der Fragebogen lediglich

als Besprechungsgrundlage für mögliche Optimierungen der Benutzungsschnittstellen genutzt.

Im Wesentlichen ergab diese Besprechung, dass die umgesetzte Benutzungsschnittstelle die Kriterien der Gebrauchstauglichkeit größtenteils erfüllt. Im Folgenden werden aus der Besprechung hervorgehende Verbesserungsvorschläge aufgeführt, welche in der zukünftigen Weiterentwicklung der Anwendung umgesetzt werden sollten:

- Bei Navigation zwischen Analyse- und Konfigurationsoberfläche sollten die Daten der Oberfläche, welche verlassen wird weiterhin ohne erneutes Laden verfügbar sein. Dafür bietet sich eine zentralisierte Zustandsverwaltung der Komponenten der Webanwendung unter Verwendung der JavaScript Bibliothek Redux an.
  - Wird auf diese Funktionalität verzichtet, so sollte zur Navigation in die jeweils andere Oberfläche ein Bestätigungsdialog erforderlich sein, in welchem dem Benutzer mitgeteilt wird, dass dies zum Verwerfen nicht gespeicherter Änderungen führt.
- Werden Anlagen in einer der beiden Oberflächen ausgewählt, so sollten diese auch in der anderen Oberfläche standardmäßig ausgewählt sein.

6.3. Evaluierung des Vorgehens

In diesem Abschnitt wird rückblickend das Vorgehen des im Rahmen der Bachelorarbeit durchgeführten Softwareentwicklungs-Prozesses evaluiert.
Die Unterteilung der Softwareentwicklung in die Phasen Analyse, Entwurf und Implementierung hat sich als sehr hilfreich herausgestellt. Insbesondere hat dies dazu geführt, dass das System auf eine Weise entwickelt wurde, welche es ermöglicht, das System mit geringem Aufwand um weitere Funktionalitäten zu ergänzen und umgesetzte Funktionalitäten dafür wiederzuverwenden.
Auch die Sammlung der umzusetzenden Aufgaben in Trello-Listen war sehr hilfreich für einen Überblick der noch anstehenden Aufgaben und um diesen Teilaufgaben in Form von Checklisten hinzuzufügen. Eine genauere Aufwandsabschätzung der einzelnen Aufgaben und ein spezifischerer Zeitplan wäre rückblickend hilfreich gewesen für ein besseres Zeitmanagement.

Aufgrund der notwendigen Einarbeitung in die zu verwendenden Datenquellen und die im Rahmen der serviceorientierten Architektur benötigten Testtechnologien wurde erst in einem späteren Stadium des Projekts automatisiert getestet. Dies führte dazu, dass während der Entwicklung manuell unter Verwendung von Konsolenausgaben, des API-Testwerkzeugs Postman und Benutzeroberfläche getestet wurde.
Frühzeitig entwickelte automatisierte Tests hätten hingegen Zeit eingespart, da diese

zum Testen der korrekten Funktionsweise aller bereits getesteten Funktionalitäten verwendet werden können, wenn neue Funktionalitäten hinzugefügt oder bestehender Quellcode angepasst wird.
Ebenfalls ist zu erwähnen, dass ein Vorgehen mittels Test-Driven-Development ebenfalls von Vorteil gewesen wäre, da dies eine sehr zielgerichtete, effektive Entwicklung zuverlässiger Software ermöglicht. Da das System jedoch ausgiebig geplant wurde und bereits vor der Implementierung der einzelnen Methoden deren Schnittstellen definiert wurden, war auch ohne Test-Driven-Development eine zielgerichtete Umsetzung möglich.

Zusammenfassend lässt sich das Vorgehen als sehr zielführend und effektiv betrachten, ausgenommen der Tatsache, dass mittels frühzeitigerem, automatisiertem Testing der in manuelle Tests investierte Aufwand geringer gewesen wäre.

7. Zusammenfassung und Ausblick

Abschließend folgen nun eine Zusammenfassung der Arbeit und ein Ausblick darauf, wie die Ergebnisse der Arbeit weiterentwickelt werden können.

7.1. Zusammenfassung

Im Rahmen dieser Bachelorarbeit wurden ein Konzept und ein Prototyp einer Webanwendung zur automatisierten Ermittlung von Optimierungsmöglichkeiten in Heizungsanlagen entwickelt. Dieses Konzept enthält eine weitsichtig geplante, jedoch entsprechend des Prinzips „keep it simple stupid“ mit gering gehaltener Komplexität umgesetzte Softwarearchitektur. Das Backend der Anwendung ist als serviceorientierte Architektur umgesetzt und so implementiert, dass insbesondere die Datenbeschaffung für weitere Funktionalitäten wiederverwendet werden kann. Zum Auslösen von Analysen, sowie zum Bearbeiten und Speichern der Analyseergebnisse in Form von Textbausteinen, wurde eine interaktive Benutzeroberfläche entwickelt.
In der Umsetzung des Systems wurde großer Wert auf das Einhalten bewährter Richtlinien gelegt und somit ein für andere Entwickler leicht verständliches, anpassbares System entwickelt.
Insbesondere erfolgte im Rahmen des Entwicklungsprozesses eine ausführliche Untersuchung geeigneter Architekturmuster, Entwurfsmuster, Technologien und Richtlinien für die Implementierung. Somit kann die schriftliche Ausarbeitung zur Orientierung

für die Software-Entwicklung in der EWUS GmbH genutzt werden.

Bei dem entwickelten System handelt es sich um einen Prototyp, für welchen im Rahmen der Bachelorarbeit nur eine kleine Auswahl an Analysen umgesetzt wurde. Dieser Prototyp ist für den produktiven Einsatz um die im Ausblick aufgeführten Funktionalitäten zu erweitern. Diese erfordern teilweise einen Lernprozess, welcher durch Verwendung des entwickelten Prototyps unterstützt wird.

7.2. Ausblick

Die im Rahmen der Bachelorarbeit entwickelten Analysen basieren auf einer vorläufigen Festlegung der Analyseparameter. Damit optimale Ergebnisse erzielt werden können, sollten sie um weitere Kriterien ergänzt und die darin enthaltenen Richtwerte angepasst werden. Das Ermitteln des dafür notwendigen Wissens kann durch Verwendung des Prototyps und der generierten Logs, über die Interaktionen mit den Analyseergebnissen, gefördert werden.
Es ist abzusehen, dass die Richtwerte der Analysen, also beispielsweise der Sollbereich des Nutzungsgrads mehrfach angepasst werden müssen, bis die entsprechenden Analysen die gewünschten Ergebnisse erzielen. Dies sollte von dem damit beauftragten Ingenieur ohne Eingriffe in den Quellcode möglich sein. Eine Möglichkeit, dies zu erreichen ist, dass die Richtwerte aus der zentralen Datenbank ausgelesen werden.

Ein weiterer zentraler Ausbauschritt des Systems ist es, weitere Analysen zu integrieren. Insbesondere kann dafür die dem Unternehmen verfügbare API zum Ermitteln der Nachtabsenkungen, Heizkurven, Heizgrenzen und Anlagenumstellungen eingebunden werden.

Zur Optimierung der Gebrauchstauglichkeit der Benutzeroberfläche sollten die in Abschnitt 6.2.8 ermittelten Verbesserungsvorschläge umgesetzt werden. Dafür ist eine zentrale Zustandsverwaltung der Komponenten des Frontends empfehlenswert.

Zur Verwendung des entwickelten System sollte dieses auf dem Server der EWUS GmbH bereitgestellt werden. Für die Bereitstellung der mit Spring Boot umgesetzten Services des Backends kann der jeweils integrierte Tomcat Server verwendet werden.

Des Weiteren kann das Backend zur vollautomatisierten Ticketerstellung weiterentwickelt werden, nachdem das für die Geschäftslogik notwendige Wissen gesammelt, integriert und umfassend getestet wurde. Anstelle der manuellen Bearbeitung und der abschließenden Speicherung der Ergebnisse in einem Energielenker-Attribut, kann

dafür „Tasks“ Ressource der Energielenker API verwendet werden. Auch die Protokoll-Erstellung im Rahmen der Anlagenoptimierung kann unter Wiederverwendung der im Backend umgesetzten Funktionalitäten automatisiert werden.

Ein weiteres denkbares Feature ist eine Prognose der Kosten und Einsparungen durch die im Rahmen der Analyse ermittelten Maßnahmen. Auch eine Analyse der Entwicklungen, welche aus den Anlagenumstellungen erfolgten und dessen Bereitstellung an die Öffentlichkeit in anonymisierter Form ist geplant. Dies kann einen wertvollen Betrag für wissenschaftliche Zwecke im Rahmen des vom Bundesministerium für Wirtschaft und Energie geförderten Projekts darstellen. Insbesondere können derartige Prognosen und Berichte ein nachhaltiges und ökologisch verantwortbares Handeln fördern.

Basierend auf dem schriftlichen Teil der Arbeit können Richtlinien für die angehenden Softwareentwickler und programmierenden Ingenieure der EWUS GmbH verfasst werden, um die Entwicklung von verständlichem, wartbaren, erweiterbaren und wiederverwendbarem Quellcode zu fördern.

Zusammenfassend kann der hier umgesetzte Prototyp als Grundgerüst für diverse weitere Analysen und Automatisierungen genutzt werden.

Quellenverzeichnis

[1] K. Arsov. *Microservices vs. SOA — Is There Any Difference at All?* In: Medium. Online: https://medium.com/microtica/microservices-vs-soa-is-there-any-difference-at-all-2a1e3b66e1be; letzter Zugriff: 13.10.2021. 2017.

[2] S. Augsten. *Was sind Microservices?* In: Dev-Insider. Online: https://www.dev-insider.de/was-sind-microservices-a-634583/; letzter Zugriff: 13.10.2021. 2017.

[3] Baeldung. *Ant vs Maven vs Gradle*. Online: https://www.baeldung.com/ant-maven-gradle; letzter Zugriff: 14.10.2021. 2020.

[4] S. Bargui. *Microservices Architektur. Definition, Chancen und Risiken*. Online: https://www.grin.com/document/542013; letzter Zugriff: 13.10.2021. 2019.

[5] M. Cneude. *A Detailed Explanation of The KISS Principle in Software*. In: The Valuable Dev. Online: https://thevaluable.dev/kiss-principle-explained/; letzter Zugriff: 14.10.2021. 2021.

[6] Software Development Community. *What Is Service-Oriented Architecture?* In: Medium. Online: https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec; letzter Zugriff: 13.10.2021. 2019.

[7] MDN Contributors. *MVC*. In: MDN Web Docs Glossary: Definitions of Web-related terms. Online: https://developer.mozilla.org/en-US/docs/Glossary/MVC; letzter Zugriff: 13.10.2021. 2021.

[8] IBM Corporation. *Kommunikationsdiagramme*. Online: https://www.ibm.com/docs/de/radfws/9.6.1?topic=SSRTLW_9.6.1/com.ibm.xtools.sequence.doc/topics/ccommndiag.html; letzter Zugriff: 14.10.2021.

[9] IBM Corporation. *Service-oriented architecture (SOA)*. Online: https://www.ibm.com/docs/en/rbd/9.5.1?topic=overview-service-oriented-architecture-soa; letzter Zugriff: 13.10.2021. 2012.

[10] IBM Cloud Education. *REST APIs*. Online: https://www.ibm.com/cloud/learn/rest-apis; letzter Zugriff: 13.10.2021. 2021.

[11] IBM Cloud Education. *SOA (Service-Oriented Architecture)*. Online: https://www.ibm.com/cloud/learn/soa; letzter Zugriff: 13.10.2021. 2021.

[12] L. Fernando. *Repository Pattern with Dependency Injection — MVC EF Code First*. In: Medium. Online: https://medium.com/aeturnuminc/repository-pattern-with-dependency-injection-mvc-ef-code-first-91344413ba1c; letzter Zugriff: 13.10.2021. 2018.

[13] R. T. Fielding. *Architectural styles and the design of network-based software architectures. Doctoral Dissertation*. Irvine, CA, USA: University of California, 2000.

[14] Energielenker GmbH. *Baum base monitor*. Online: intranet/ewus.elmonitor.de/inventar/tree#; letzter Zugriff: 13.10.2021.

[15] SMF GmbH. *Architektur und Entwurfsmuster. Definition, Abgrenzung und ausgewählte Beispiele*. Online: https://www.smf.de/pdf/Architektur_und_Entwurfsmuster_2012.pdf; letzter Zugriff: 13.10.2021. 2012.

[16] t2informatik GmbH. *User Story*. Online: https://www.microtool.de/wissen-online/was-ist-eine-user-story/; letzter Zugriff: 13.10.2021.

[17] Webrunners GmbH. *Web vs. Desktop-Anwendungen – Was ist die bessere Wahl?* Online: https://www.webrunners.de/web-vs-desktop-anwendungen-was-ist-die-bessere-wahl/; letzter Zugriff: 13.10.2021. 2020.

[18] J. Goll und M. Dausmann. *Architektur- und Entwurfsmuster der Softwaretechnik. Mit lauffähigen Beispielen in Java*. 1. Auflage. Wiesbaden: Springer Vieweg, 2013. ISBN: 978-3-8348-2432-5.

[19] T. Hamilton. *Unit Test vs Integration Test: What's the Difference?* Online: https://www.guru99.com/unit-test-vs-integration-test.html; letzter Zugriff: 13.10.2021. 2021.

[20] M. Harsh. *Ant Design: The World's 2nd Most Popular React UI Library*. In: Medium. Online: https://medium.com/codex/ant-design-the-worlds-2nd-most-popular-react-ui-library-b6c4853aefaa; letzter Zugriff: 13.10.2021. 2021.

[21] Heizungsbau.net. *Heizungsanlage nachrüsten – Brennwertheizung oder Niedertemperaturkessel?* Online: https://www.heizungsbau.net/magazin/brennwertheizung-niedertemperaturkessel-20152359; letzter Zugriff: 13.10.2021. 2021.

[22] *IEEE Standard Glossary of Software Engineering Terminology*. New York, NY, USA: Institute of Electrical und Electronics Engineers (IEEE), 1990. ISBN: 1-55937467-X.

[23] Facebook Inc. *Introducing Hooks*. Online: https://reactjs.org/docs/hooks-intro.html; letzter Zugriff: 14.10.2021. 2021.

[24] Red Hat Inc. *Microservices. Das Service Mesh – Funktionsweise und Vorteile*. Online: https://www.redhat.com/de/topics/microservices/what-is-a-service-mesh; letzter Zugriff: 13.10.2021. 2021.

[25] Red Hat Inc. *Was sind Microservices?* Online: https://www.redhat.com/de/topics/microservices/what-are-microservices; letzter Zugriff: 13.10.2021. 2021.

[26] Red Hat Inc. *What is a REST API?* Online: https://www.redhat.com/en/topics/api/what-is-a-rest-api; letzter Zugriff: 13.10.2021. 2020.

[27] VMWare Inc. *Spring Boot*. Online: https://spring.io/projects/spring-boot; letzter Zugriff: 14.10.2021. 2021.

[28] VMWare Inc. *Spring Data JPA*. Online: https://spring.io/projects/spring-data-jpa; letzter Zugriff: 14.10.2021. 2021.

[29] ITwissen.info. *Java persistence API (JPA)*. Online: https://www.itwissen.info/JPA-Java-persistence-API.html; letzter Zugriff: 14.10.2021. 2020.

[30] S. Janser. *Eureka – Microservice-Registry mit Spring Cloud*. In: Heise Developer. Online: https://www.heise.de/developer/artikel/Eureka-Microservice-Registry-mit-Spring-Cloud-2848238.html; letzter Zugriff: 13.10.2021. 2015.

[31] T. Janssen. *Implementing the Repository pattern with JPA and Hibernate*. Online: https://thorben-janssen.com/implementing-the-repository-pattern-with-jpa-and-hibernate/; letzter Zugriff: 13.10.2021.

[32] A. Kee. *Introduction to client-server architecture (frontend backend)*. Online: https://fluffy.es/introduction-to-client-server/; letzter Zugriff: 13.10.2021. 2019.

[33] LeanIX. *Der ultimative Guide zu Microservices-Architekturen*. Online: https://www.leanix.net/de/microservices-architecture; letzter Zugriff: 13.10.2021.

[34] B. Legierski. *Frontend vs Backend*. Online: https://www.evertop.pl/en/frontend-vs-backend/; letzter Zugriff: 13.10.2021. 2021.

[35] F. Lindner. *Serviceorientierte Architektur*. Online: https://dewiki.de/Lexikon/Serviceorientierte_Architektur; letzter Zugriff: 13.10.2021.

[36] S. Luber und A. Donner. *Was ist das Client-Server-Modell?* In: Dev-Insider. Online: https://www.ip-insider.de/was-ist-das-client-server-modell-a-940627/; letzter Zugriff: 13.10.2021. 2020.

[37] Lucidchart. *What is Unified Modeling Language*. Online: https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language; letzter Zugriff: 14.10.2021.

[38] R. C. Martin und J. O. Coplien. *Clean Code. A Handbook of Agile Software Craftmanship*. 1. Auflage. Upper Saddle River, NJ, USA: Prentice Hall, 2008. ISBN: 978-0-13-235088-4.

[39] Medium. *6 Most Popular Java Web Frameworks in 2021*. Online: https://medium.com/@tech_multi/6-most-popular-java-web-frameworks-in-2021-c66eb7d27a6; letzter Zugriff: 14.10.2021. 2021.

[40] H. Mu und S. Jiang. *Design patterns in software development*. In: 2011 IEEE 2nd International Conference on Software Engineering and Service Science. Institute of Electrical und Electronics Engineers (IEEE), 2016. ISBN: 978-1-4244-9698-3.

[41] R. Paschotta. *Heizkessel*. Online: https://www.energie-lexikon.info/heizkessel.html; letzter Zugriff: 13.10.2021. 2020.

[42] R. Paschotta. *Nutzungsgrad*. In: RP-Energie-Lexikon. Online: https://www.energie-lexikon.info/nutzungsgrad.html; letzter Zugriff: 13.10.2021. 2021.

[43] R. Paschotta. *Vorlauftemperatur*. Online: https://www.energie-lexikon.info/vorlauftemperatur.html; letzter Zugriff: 13.10.2021. 2021.

[44] Peters, J. et al. *Architectural Pattern Definition for Semantically Rich Modular Architectures*. In: 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA). Institute of Electrical und Electronics Engineers (IEEE), 2016. ISBN: 978-1-5090-2131-4.

[45] D. Rajput. *Hands-On Microservices - Monitoring and Testing*. In: Packt. Online: https://www.packtpub.com/product/hands-on-microservices-monitoring-and-testing/9781789133608; letzter Zugriff: 20.10.2021.

[46] G. Rauch und S. Augsten. *Was ist Spring Boot?* In: Dev-Insider. Online: https://www.dev-insider.de/was-ist-spring-boot-a-1009135/; letzter Zugriff: 14.10.2021. 2021.

[47] M. Rehkopf. *What is a kanban board?* In: Atlassian Agile Coach. Online: https://www.atlassian.com/agile/kanban/boards; letzter Zugriff: 13.10.2021.

[48] C. Richardson. *Pattern: Service registry*. In: microservices.io. Online: https://microservices.io/patterns/service-registry.html; letzter Zugriff: 13.10.2021. 2021.

[49] M. Rohde. *Echtdaten sind keine Testdaten – Was bei Softwaretests zu beachten ist*. Online: https://www.datenschutz-notizen.de/echtdaten-sind-keine-testdaten-was-bei-softwaretests-zu-beachten-ist-5918937/; letzter Zugriff: 14.10.2021. 2017.

[50] RubyGarage. *Best Architecture for an MVP: Monolith, SOA, Microservices, or Serverless?* Online: https://rubygarage.org/blog/monolith-soa-microservices-serverless; letzter Zugriff: 13.10.2021. Apr. 2019.

[51] T. Samarpit. *Microservices vs SOA: What's the Difference?* In: DZone. Online: https://dzone.com/articles/microservices-vs-soa-whats-the-difference; letzter Zugriff: 20.10.2021. 2018.

[52] I. Sommerville. *Software Engineering*. 8. Auflage. Harlow, England: Addison-Wesley, 2006. ISBN: 978-0321313799.

[53] K. Stoll. *Was ist eigentlich React?* In: t3n - digital pioneers. Online: https://t3n.de/news/react-facebook-623999/; letzter Zugriff: 14.10.2021. 2021.

[54] Effiziente Wärme und Stromlieferung GmbH. *Einsparzähler-Programm Projekt Digitaler Heizraum*. Online: https://www.ewus.berlin/energiemanagement/expertensystem-digitaler-heizraum/; letzter Zugriff: 13.10.2021. Apr. 2021.

[55] K. Wickramanayake. *Is MVC a design pattern or an architectural pattern?* Online: http://www.swview.org/blog/mvc-design-pattern-or-architectural-pattern; letzter Zugriff: 13.10.2021. 2010.

[56] Willumeit, H. et al. *IsoMetrics S. Fragebogen zur Evaluation von graphischen Benutzungsschnittstellen*. 1993.

Abkürzungsverzeichnis

Abkürzung	**Bedeutung**
API	Application programming interface
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JPA	Java Persistance API
KISS	Keep it simple stupid
MVC	Model View Controller
URL	Uniform Resource Identifier
SOA	Serviceorientierte Architektur
SQL	Structured Query Language
UML	Unified Modeling Language

A. Appendix

A.1. Quellcode und Demonstrationsvideo

Der im Rahmen der Bachelorarbeit entwickelte Quellcode ist gemeinsam mit einem Demonstrationsvideo in der HTW Cloud bereitgestellt. Darüber hinaus wurden beide Betreuer in das private, auf Github gehostete Repository, welches zur Versionsverwaltung genutzt wurde, eingeladen. Die Abgabe befindet sich dort in dem Branch „master“, welchem bis zum Abschluss der Bewertung keine weiteren Änderungen hinzugefügt werden. Das Repository ist über den folgenden Link abrufbar: `https://github.com/janis-schanbacher/bachelorarbeit`

A.2. Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

_____

Ort, Datum, Unterschrift