

DIPLOMARBEIT

Visualisierung der Ergebnisse des Stromnetzmodells

Ausgeführt im Schuljahr 2023/24 von:

Betreuer:

Felix Schneider

5AHIT-19

Ing. Jürgen Katzenschlager MSc, BEd

Clemens Schlipfänger

5AHIT-17

Ing. Jürgen Katzenschlager MSc, BEd

Krems, am 02. April 2024

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Krems, am 02. April 2024

Verfasser/innen:

Felix Schneider

Clemens Schlipfinger

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT
	Krems
	Abteilung: Informationstechnologie

DIPLOMARBEIT

DOKUMENTATION


Namen der Verfasser/innen	Clemens Schlipfnger Felix Schneider
Jahrgang / Klasse Schuljahr	5AHIT 2023/24
Thema der Diplomarbeit	Visualisierung der Ergebnisse des Stromnetzmodells
Kooperationspartner	Siemens AG Österreich

Aufgabenstellung	Siemens entwickelt ein neues Programmpaket zur Echtzeitberechnung von Stromnetzwerken. Für diese Berechnung ist die Qualität des Netzmodells von höchster Wichtigkeit, aber aufgrund seiner Größe sind Fehler unvermeidlich. Aktuell werden Fehler in unübersichtlichen Log Files gespeichert. Ein ausfallsicheres System mit strukturierter Visualisierung wird für einfache Auswertungen benötigt.
------------------	--

Realisierung	Die Applikation verwendet für das <i>Backend</i> die Technologien „Apache Kafka“, „PostgreSQL“ und das „Java Spring“- <i>Framework</i> . Die Schnittstelle zum <i>Frontend</i> ist mit „GraphQL“ implementiert worden und das <i>Frontend</i> selbst mit „Angular“.
--------------	---

Ergebnisse	Das Projektziel ist die Entwicklung einer benutzerfreundlichen Webanwendung mit Filtermöglichkeiten, Graphen und Diagrammen. Dabei wird ein Backend-System verwendet, welches durch die Verwendung von Apache Kafka reibungslos in die Siemens-Infrastruktur integriert werden kann und gleichzeitig höchste Stabilität und Fehlertoleranz gewährleistet. Dadurch wird es den Siemens-Ingenieur:innen erleichtert, Netzmodellfehler zu analysieren.
------------	---

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

Architektur der Applikation	<div data-bbox="643 342 1422 680">  </div> <p data-bbox="643 689 1422 880">Diese Grafik zeigt den Architekturaufbau unseres Prototypen. Die Daten werden vom System von Siemens AG über Apache Kafka in die PostgreSQL Datenbank gespeichert. Über die GraphQL API Schnittstelle stehen diese Daten dem Frontend zur Verfügung.</p>
-----------------------------	---

Teilnahme an Wettbewerben, Auszeichnungen	Bosch Innovationspreis 2024
---	-----------------------------

Möglichkeiten der Einsichtnahme in die Arbeit	Der schriftliche Teil der Arbeit ist öffentlich einsehbar. Jedoch gibt es einen Sperrvermerk auf den Prototypen, da Siemens das Copyright auf den Programmcode besitzt.
---	---

Approbation (Datum / Unterschrift)	Prüfer/in	Abteilungsvorstand / Direktor/in
---------------------------------------	-----------	-------------------------------------

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems Abteilung: Informationstechnologie
-------------------	--

DIPLOMA THESIS

Documentation


Authors	Clemens Schlipfänger Felix Schneider
Form Academic year	5AHIT 2023/24
Topic	Visualisation of the results of the electricity grid model
Co-operation partners	Siemens AG Austria

Assignment of tasks	Siemens is developing a new programme package for the real-time calculation of power grids. The quality of the network model is of the utmost importance for this calculation, but due to its size, errors are unavoidable. Currently, errors are stored in confusing log files. A fail-safe system with structured visualisation is required for simple evaluations.
---------------------	---

Realization	The application uses the technologies „Apache Kafka“, „PostgreSQL“ and the „Java Spring“-Framework for the backend. The interface to the frontend has been implemented with „GraphQL“ and the frontend itself with „Angular“.
-------------	---

Results	The project goal is to develop a user-friendly web application with filtering capabilities, graphs, and charts. This will involve utilizing a backend system that seamlessly integrates into the Siemens infrastructure by leveraging Apache Kafka, while ensuring the highest level of stability and fault tolerance. This will facilitate Siemens engineers in analyzing network model errors.
---------	--

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

Architecture of the application	<div data-bbox="643 338 1418 678">  </div> <p data-bbox="643 689 1428 835">This graphic shows the architecture of our prototype. The data comes from the Siemens GNA and is stored in the PostgreSQL database through Kafka. From there, it can be retrieved by the frontend via a GraphQL API.</p>
---------------------------------	---

Participation in competitions Awards	Bosch Innovation Award 2024
--------------------------------------	-----------------------------

Accessibility of diploma thesis	The written part of the work is publicly accessible. However, there is a blocking notice on the prototypes, as Siemens owns the copyright to the programme code.
---------------------------------	--

Approval (Date / Sign)	Examiner	Head of Department / College
---------------------------	----------	---------------------------------

Inhaltsverzeichnis

1. Präambel	10
1.1. Team	10
1.2. Kurzfassung	10
1.3. Abstract	10
1.4. Danksagung	11
1.5. Gendererklärung	11
2. Einleitung	12
2.1. Ausgangssituation und Problemstellung	12
2.2. Initial situation and problem definition	13
2.3. Forschungsfragen	14
2.3.1. Message Propagation	14
2.3.2. Optimale Darstellungsmethoden	14
2.4. Strukturierung der Arbeit	14
3. Message Propagation	15
3.1. Verteilte Systeme in der Service-Oriented Architecture	15
3.1.1. Eigenschaften zur Bewertung von verteilten Systeme	15
3.1.2. Kommunikationstechniken in verteilte Systeme	17
3.2. Enterprise Messaging Systems	20
3.2.1. Enterprise Service Bus	21
3.2.2. Nachrichten	22
3.2.3. Queues	22
3.2.4. Message Broker und Cluster	23
3.2.5. Messaging Models	24
3.2.6. Consumption Mode	25
3.2.7. Quality-of-Service Garantien	26
3.2.8. Protokoll für die Kommunikation mit dem Broker	27
3.2.9. Latenz und Durchsatz	28
3.3. Vergleich von populären Enterprise Messaging Services	28
3.3.1. Apache Kafka	29
3.3.2. RabbitMQ	32
4. Graphentheorie	34
4.1. Grundlagen eines Graphen	34
4.2. Ausrichtung	34
4.3. Konnektivität	34
4.4. Zyklen	35
4.5. Gewicht der Kanten	36
4.6. Vollständigkeit	36
4.7. Bipartition	37
4.8. Planarität	37
4.9. Eulerscher Graph	38
4.10. Hamiltonscher Graph	39

5. Visualisierung	40
5.1. Benutzerfreundlichkeit und Performance	40
5.1.1. Intuitivität	40
5.1.2. Interaktivität	41
5.1.3. Performance	42
5.2. Arten von Datendarstellungen	42
5.2.1. Darstellung von einfachen Daten in Tabellen, Diagrammen und Kennzahlen	43
5.2.2. Darstellung von komplexen Daten in Graphen, Heatmaps und Diagrammen	45
5.3. Bibliothek zur visuellen Darstellung insbesondere von Graphen	50
6. Architektur des Prototypen	52
6.1. Apache Kafka und das Backend	52
6.1.1. Messaging Service: Apache Kafka	52
6.1.2. Logik im Backend: Spring Framework	53
6.1.3. Persistierung der Daten: PostgreSQL Datenbank	55
6.2. GraphQL und das Frontend	57
6.2.1. Die Schnittstelle zwischen Backend und Frontend: GraphQL	57
6.2.2. Das Framework aus der JavaScript Welt: Angular	60
6.2.3. Die Bibliothek zum Erstellen des Graphes: Cytoscape	62
7. Backend-System	64
7.1. Apache Kafka und Java Spring Framework	64
7.1.1. Einrichtung von Apache Kafka	64
7.1.2. Verwendung von Apache Kafka im Java Spring Framework	65
7.2. PostgreSQL	67
7.2.1. Problematik der Speicherung von Graph-Daten in einer relationalen Datenbank	67
7.2.2. Lösungsansatz für effiziente Speicherung von Graph-Daten	68
8. Webapplikation Angular	70
8.1. Dashboard mit Kennzahlen des Netzmodells	70
8.2. Tabellen mit Filter- und Suchfunktionen	72
8.2.1. Sortierung	73
8.2.2. Filterung	74
8.2.3. Paginierung	75
8.2.4. Besonderheit: Spannung	75
8.3. Graph des Stromnetzmodells	76
8.4. Applikation im Dark Mode	79
9. Bewertung	81
9.1. Enterprise Service Bus	81
9.2. Visualisierung des Netzmodells	81
10. Zusammenfassung und Ausblick	82
10.1. Zusammenfassung	82
10.2. Ausblick	82

I. Literaturverzeichnis	83
II. Abbildungsverzeichnis	86
III. Tabellenverzeichnis	88
IV. Quellcodeverzeichnis	89
V. Abkürzungsverzeichnis	90
VI. Übersetzungsverzeichnis	91
A. Anhang	92
A.1. Kapitelverzeichnis	92
A.2. Besprechungsprotokolle	93
A.2.1. Begleitprotokoll 1	93
A.2.2. Begleitprotokoll 2	96
A.2.3. Begleitprotokoll 3	99
A.2.4. Begleitprotokoll 4	102
A.2.5. Begleitprotokoll 5	105
A.3. Pflichtenheft	108
A.4. Arbeitspakete	118
A.5. Projektstagebücher	119
A.5.1. Projektstagebuch Clemens Schlipfinger	119
A.5.2. Projektstagebuch Felix Schneider	121
A.6. Datenträgerbeschreibung	125
A.7. Ausführung des Prototypen	125

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

1. Präambel

1.1. Team

An dieser Arbeit sind einige Personen involviert. In erster Linie haben Felix Schneider und Clemens Schlipfinger die Arbeit verfasst. Eine große Unterstützung bei diesem Projekt ist unser Betreuer Ing. Jürgen Katzenschlager MSc, BEd.

Das Projektteam besteht aus dem Projektleiter Felix Schneider und Kollegen Clemens Schlipfinger. Betreuer des Projekts ist Ing. Jürgen Katzenschlager MSc, BEd und Auftraggeber ist Siemens AG Österreich.

Der Kontakt mit Siemens wurde über den Ingenieur Johannes Kurz abgehalten, der dem Projektteam ständig für Fragen zur Verfügung stand.

1.2. Kurzfassung

Siemens entwickelt ein neues Programmpaket zur Echtzeitberechnung von Stromnetzwerken. Für diese Berechnung ist die Qualität des Netzmodells von höchster Wichtigkeit, aber aufgrund seiner Größe sind Fehler unvermeidlich. Aktuell werden Fehler in unübersichtlichen Protokolldateien gespeichert. Ein ausfallsicheres System mit strukturierter Visualisierung wird für einfache Auswertungen benötigt.

1.3. Abstract

The Austrian company Siemens AG is developing a new application for the real-life calculation of the domestic and international utility grid within Europe. The quality of this electricity system is very important for these calculations. However, the immense size necessarily leads to errors within the system. These failures are currently tracked in simple text files / log files. Therefore, the need for a fail-safe system with intuitive visualizations arises.

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

1.4. Danksagung

Unser besonderer Dank gilt Ing. Jürgen Katzenschlager MSc, BEd, unserem Betreuer, der uns während des gesamten Entwicklungsprozesses unserer Arbeit mit seiner Fachkompetenz unterstützte und uns wertvolle Ratschläge gab. Die regelmäßigen Meetings, dessen Begleitprotokolle auch im Anhang einsehbar sind, sind immer reibungslos und aufklärend verlaufen. Darüber hinaus haben wir von genannter, inspirierender Persönlichkeit in den Unterrichtsstunden „Einführung in das wissenschaftliche Arbeiten“ wöchentliche Motivationstipps und generelle Lösungsvorschläge erhalten. Danke für alles!

Wir möchten uns auch Johannes Kurz bedanken, der als unser Ansprechpartner die Kooperation mit der Siemens AG Österreich ermöglichte. Seine großzügige Unterstützung zeigte sich nicht nur darin, dass er sich persönlich Zeit für uns nahm, sondern auch darin, dass er uns sogar einen exklusiven Durchgang durch das Siemens Hauptgebäude in Wien Floridsdorf ermöglichte. Sein Engagement und seine Hilfsbereitschaft haben unsere Arbeit enorm bereichert und waren von unschätzbarem Wert für unser Projekt.

Des Weiteren möchten wir an dieser Stelle unseren Familien und Freunden unseren aufrichtigen Dank aussprechen, die uns in vielfältiger Weise bei unserem Vorhaben unterstützt haben. Ihre bedingungslose Unterstützung erstreckte sich über verschiedene Aspekte, angefangen von einem offenen Ohr für unsere Anliegen bis hin zu emotionaler Unterstützung und aufmerksamen Zuhören, wenn wir über die Herausforderungen und Bedenken im Zusammenhang mit unserem Projekt sprachen. Durch ihre Präsenz und ihr Verständnis haben sie maßgeblich dazu beigetragen, die emotionalen Belastungen, die mit unserem Projekt einhergingen, effektiv zu bewältigen. Ihre Unterstützung war für uns von unschätzbarem Wert und hat entscheidend zum erfolgreichen Abschluss unseres Vorhabens beigetragen.

1.5. Gendererklärung

In der vorliegenden Diplomarbeit wurde die männliche Form als Sprachkonvention gewählt, um die Lesbarkeit zu erleichtern. Dies bedeutet, dass Begriffe wie „Benutzerinnen und Benutzer“ aus Gründen der Textverständlichkeit auf die kürzere Form „Benutzer“ reduziert wurden. Diese Vereinfachung der Sprache soll jedoch keineswegs eine Geschlechterdiskriminierung implizieren, sondern lediglich eine klare und verständliche Darstellung des Textes gewährleisten.

2. Einleitung

2.1. Ausgangssituation und Problemstellung

Das Unternehmen Siemens entwickelt für viele Kunden innerhalb und außerhalb von Österreich ein System, welches die Ausfallsicherheit des Stromnetzwerkes ständig überprüft und somit garantiert. Diese Software trägt den Namen *Siemens GNA*. Aufgrund der unzähligen Elemente, wie zum Beispiel Sammelschienen, Ableiter, Generatoren, Transformatoren, Trennschalter, Leistungsschalter, Stationen, Sicherungen und Lasttrennschalter, besteht dieses Stromnetzwerk aus äußerst komplexen Daten. Außerdem kann es bei so vielen Elementen leicht passieren, dass gewisse Fehler, meistens in Form von Abweichungen von Sollwerten, im Netzmodell auftreten. Diese Fehler werden von Siemens erfasst, jedoch wird aktuell über keine Applikation verfügt, welche diese gefundenen Fehlerdaten effizient verarbeitet und visualisiert. Die Ingenieure bei Siemens analysieren die Fehler mittels einfachen Textdateien, welche nur schwer lesbar sind und bei einem groben Ausfall die Dauer der Reparatur unnötig vergrößern.

Um die Analyse der Fehlerdaten effizienter zu machen, schreiben Clemens Schlipfinger und Felix Schneider eine Applikation, welche diese stark vernetzten Daten mit optimalen Darstellungsarten visualisiert. Dabei gehen wir in dieser Arbeit besonders auf die Gestaltung eines entkoppelten Backend-Systems und einige geeigneten Visualisierungsarten für solch komplexe Daten ein.

In der Abbildung 2.1 wird die Architektur unserer Applikation zur Fehlervisualisierung dargestellt. Die Fehlerdaten werden von der Siemens GNA Software (*Global Network Analysis*) erzeugt und mittels Apache Kafka dem Backend-System übertragen. Damit wird eine hohe Entkoppelung zwischen der Siemens Software und unserem System erreicht. Für die Entwicklung und die Verwaltung von beiden Systemen ist eine hohe Unabhängigkeit von großem Wert. Anschließend werden diese Daten in eine PostgreSQL Datenbank gespeichert und mit einer GraphQL API zur Verfügung gestellt. Das Frontend, welches mit dem JavaScript Framework Angular entwickelt worden ist, wird mit Tabellen und Graphen die Fehler visualisieren.



Abbildung 2.1.: Diese Darstellung zeigt den schematischen Aufbau der Applikation.

2.2. Initial situation and problem definition

The company Siemens develops a system called Siemens GNA for many customers both within and outside of Austria, which constantly monitors and guarantees the reliability of the power grid. Due to the numerous elements such as busbars, surge arresters, generators, transformers, disconnectors, circuit breakers, stations, fuses, and load break switches, this power grid consists of highly complex data. Additionally, with so many elements, it is easy for certain errors, usually in the form of deviations from set values, to occur in the network model. These errors are detected by Siemens, but currently, there is no application available to efficiently process and visualize this collected error data.

Engineers at Siemens analyze the errors using simple text files, which are difficult to read and unnecessarily prolong the duration of repair in the event of a major outage. To make the analysis of error data more efficient, Clemens Schlipfing and Felix Schneider are developing an application that visualizes these interconnected data with optimal visualization methods. In this work, they focus particularly on the design of a decoupled backend system, which will be implemented in the prototype using Apache Kafka, and some suitable visualization methods for such complex data.

As you can see, the illustration 2.2 visualizes the architecture of our system. First, the fault data is generated by a piece of software from Siemens, which is called GNA (*Global Network Analysis*) and transported to the backend over a fail-safe Apache Kafka system. This message bus decouples our system and the software from Siemens even more and enhances the reliability. Subsequently, the data is saved into a relational database called PostgreSQL. The GraphQL API provides an interface for the frontend application, which is based on the JavaScript-Framework Angular. The frontend offers great visualizations and filters in order to provide intuitive information.



Abbildung 2.2.: This illustration shows the architecture of our application.

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

2.3. Forschungsfragen

2.3.1. Message Propagation

Die Ausfallsicherheit moderner Systeme ist von höchster Wichtigkeit. Entkoppelte Systeme fördern die Last, unter welcher Applikationen operieren können. Ein Service ist mit anderen niedrig gekoppelt, wenn wenig Abhängigkeiten gegeben sind. Die Messagepropagation unterstützt ein System dahingehend, die Kommunikation zwischen den einzelnen Softwarekomponenten in ein eigenes Service auszulagern. Welche Form der Messagepropagation am besten für verschiedene Anwendungszwecke geeignet ist, ist demnach eine Frage von hoher Bedeutung.

Demnach wird folgendes Gebiet erforscht:

Vergleich der verschiedenen Formen der Messagepropagation in Enterprise Service Bus Technologien

2.3.2. Optimale Darstellungsmethoden

Soziale Netzwerke, biologische Systeme, Nahrungsketten, Dateisysteme und viele weitere Vorkommnisse stark vernetzter Daten verlangen eine effiziente Visualisierung dieser Datenflut. Vernetzungen dieser Art können in den meisten Fällen in gigantischen Graphen dargestellt werden, jedoch können daraus keine Informationen für das Treffen relevanter Entscheidungen getroffen werden. Wie komplexe Daten deswegen intuitiv und verständlich dargestellt werden, ist eine essenzielle Frage bei der Entwicklung solcher Systeme.

Daraus ergibt sich folgende Forschungsfrage:

Visualisierungsmethoden für stark vernetzte Daten

2.4. Strukturierung der Arbeit

Der Inhalt dieser Arbeit unterteilt sich in drei Hauptabschnitte. Jedes Kapitel ist eindeutig einem Abschnitt zuordenbar. Die Hauptabschnitte beschäftigen sich in erster Linie mit dem Sammeln aller aktuell vorhandenen Informationen zu den jeweiligen Themen, bekannt als *State of the Art* oder *Research*. Um mit der Materie vertraut zu werden, werden bestehende Forschungen der Literatur zusammengetragen und aufgearbeitet. Anschließend folgt der empirische Teil mit den Erklärungen der Implementierung des Prototyps, welcher zum Beweisen der These herangezogen wird, und schlussendlich die Bewertung der Forschungsfragen. Letzteres inkludiert einen Katalog zur Bestimmung der richtigen Form von Message Propagation und Vor- und Nachteile der verschiedenen Visualisierungsmethoden bei stark vernetzten Daten erstellt.

3. Message Propagation

Um eine hohe Flexibilität, unabhängige Entwicklung und eine bessere Verteilung der Last zu erreichen, ist die Unabhängigkeit zwischen der Siemens GNA Software und dem Backend von großer Bedeutung. Durch die Aufteilung des Systems in kleine Teile entsteht ein verteiltes System in der *Service-Oriented Architecture*. Jedoch ist die Koordination und die Synchronisation solcher Systeme äußerst komplex und verlangt hohen Aufwand. Mit dieser Problematik befasst sich die *Message Propagation* und bietet eine Reihe an Lösungsansätze, um diese Komplexität zu vereinfachen. In dieser Arbeit wurde sich für die Technologie der *Messaging Systeme* entschieden, die es verschiedenen Teilen eines verteilten Systems ermöglicht, unabhängig voneinander zu kommunizieren. Dadurch kann die entwickelte Applikation eine starke Entkopplung, Skalierbarkeit und eine schnelle Echtzeitverarbeitung gewährleisten.

Im nächsten Abschnitt werden die verteilten Systeme der *Service-Oriented Architecture* mit Fokus auf der internen Kommunikation erklärt. Die darauf folgenden Kapitel werden die Eigenschaften und Struktur von *Enterprise Messaging Systemen* vertiefen. Des Weiteren werden zwei populäre Implementierungen dieser Technologie, nämlich *RabbitMQ* und *Apache Kafka*, vorgestellt.

3.1. Verteilte Systeme in der Service-Oriented Architecture

In der *Service-Oriented Architecture* (SOA) werden Softwareressourcen als *Services* verpackt, die eigenständige Module darstellen. Diese bieten standardisierte Geschäftsfunktionalitäten und sind unabhängig vom Zustand oder Kontext anderer Dienste. Es wird eine Sammlung von *Services* erstellt, die miteinander kommunizieren können, indem sie Service-Schnittstellen verwenden, um Nachrichten von einem *Service* an ein anderes zu übermitteln oder eine Aktivität zwischen zwei oder mehreren Teilen des Systems zu koordinieren. In der Abbildung 3.1 ist ein einfacher Aufbau mit den Abhängigkeiten der einzelnen Module eines SOAs zu erkennen. Auf diese Weise kann mit SOA ein sehr flexibles, entkoppeltes System geschaffen werden. Jedoch ist die Koordination der *Services* eine große Herausforderung. Dadurch sind einige Kommunikationstechniken entstanden, welche im Kapitel 3.1.2 beschrieben sind. Des Weiteren weisen verteilte Systeme Eigenschaften auf, an denen eine Bewertung des Systems möglich ist, diese sind im Kapitel 3.1.1 erläutert. [1]

3.1.1. Eigenschaften zur Bewertung von verteilten Systeme

Die Entwicklung eines verteilten Systems ist kein leichtes Unterfangen, dadurch ist es sehr wichtig, die Funktionsfähigkeit des Anwendungssystems ermitteln zu können. In der folgenden Liste sind die Eigenschaften, welche eine leichte Bewertung ermöglichen, mit ihren Erklärungen aufgezählt.

[2, 3]

- **Kopplung:** Eine hohe Entkopplung der einzelnen Komponenten spielt eine wichtige Rolle, um eine unabhängige Entwicklung und eine hohe Flexibilität der Anwendung zu erreichen.



Abbildung 3.1.: Darstellung eines Systems in der *Service-Oriented Architecture*

- **Latenz:** Da in verteilten Systemen die Ressourcen aufgeteilt werden, ist es von Bedeutung, die Kommunikation der Komponenten mit einer niedrigen Verzögerung zu gewährleisten.
- **Skalierbarkeit:** Es ist von essenzieller Bedeutung, dass moderne Anwendungen in der Lage sind, sich an die Belastung anzupassen, die durch die Interaktion der Benutzer mit der Software entsteht. Dies ermöglicht einen reibungslosen Ablauf.
- **Wartbarkeit:** Verteilte Systeme müssen in der Lage sein, ihre Komplexität zu abstrahieren und dadurch eine leichte Wartbarkeit zu realisieren. Denn nicht überschaubare Systeme stellen langfristig einen erheblichen Kostenfaktor in der Entwicklung und Wartung dar.
- **Zuverlässigkeit:** Um eine erfolgreiche Anwendung zu gewährleisten, bedarf es einer Software mit hoher Robustheit und Zuverlässigkeit, da Fehler in der Anwendung unerwünschte Auswirkungen haben können, die in Produktivsystemen nicht toleriert werden können.
- **Verfügbarkeit:** Wenn es zu Ausfällen von Anwendungssystemen kommt, können erhebliche Kosten in Millionenhöhe entstehen. Daher ist es von entscheidender Relevanz, diese Ausfälle zu minimieren.
- **Erweiterbarkeit:** Im Lebenszyklus einer Anwendung ist die kontinuierliche Erweiterbarkeit tief verankert, da sich Geschäftsszenarien ständig ändern können. Eine Software, die die Fähigkeit besitzt, sich flexibel an diese Anforderungen anzupassen, bietet erhebliche Vorteile.

3.1.2. Kommunikationstechniken in verteilte Systeme

Die Koordination und Kommunikation in verteilten Systemen können schnell zu einer äußerst anspruchsvollen Aufgabe werden, sobald das System eine bestimmte Größe erreicht. Man muss sich nur vorstellen, wie viele kleine *Services* ein Banking-System von Unternehmen wie Sparkasse oder Raiffeisen haben muss, welche alle koordiniert und synchronisiert werden müssen. Es ist offensichtlich, dass solche Systeme leicht zu einem unüberschaubaren Netzwerk von Abhängigkeiten werden können. Ein weiterer Punkt, welcher zu beachten ist, wäre: Was passiert, wenn ein *Service* ausfällt und damit der Zugriff auf eine Ressource nicht mehr möglich ist? Eine Ausnahmesituation dieser Art kann leicht zu Inkonsistenzen im System führen oder im Extremfall eine Kettenreaktion auslösen, die zu einem totalen Ausfall der gesamten Anwendung führt. Damit solche Szenarien vermieden werden können und der Informationsaustausch der Module des verteilten Systems verlässlich und einfach gestaltet werden kann, haben sich verschiedene Kommunikationstechniken etabliert, welche in den nachfolgenden Unterkapiteln erklärt und verglichen werden.

3.1.2.1. Direkte Kommunikation zwischen den Services

Die trivialste Lösung wäre eine Form von *Remote Procedure Calls (RPC)* zwischen den einzelnen Teilen des verteilten Systems. Ein RPC ist eine Technik, die es einem Programm ermöglicht, eine Funktion oder Methode in einem anderen entfernten Programm aufzurufen, als ob sie lokal aufgerufen würde. Allerdings führt dies zu einer hohen Kopplung, da die einzelnen *Services* direkt miteinander kommunizieren, wie es in der Darstellung 3.2 zu erkennen ist. Die Natur dieser Kommunikationstechnik benötigt eine synchrone Übertragung, weil die Kommunikationspartner stets auf die Rückmeldung des anderen warten müssen, um Gewissheit zu erlangen, dass die Nachricht erfolgreich übermittelt wurde. Die synchrone Übertragung hat eine geringere Skalierbarkeit im Gegensatz zur asynchronen Übertragung, da durch das Warten auf die Rückmeldung Verzögerungen entstehen. Wenn diese Systeme eine gewisse Größe erreichen, werden sie zu undurchsichtigen Netzen an Abhängigkeiten, was zu aufwendiger Wartung und erschwerter Erweiterung des Systems führt. Dennoch können die Anforderungen an die Echtzeitkommunikation erfüllt werden, da alle *Services* direkt miteinander kommunizieren, was Verzögerungen durch Mittelsmänner vermeidet. [3, 2]

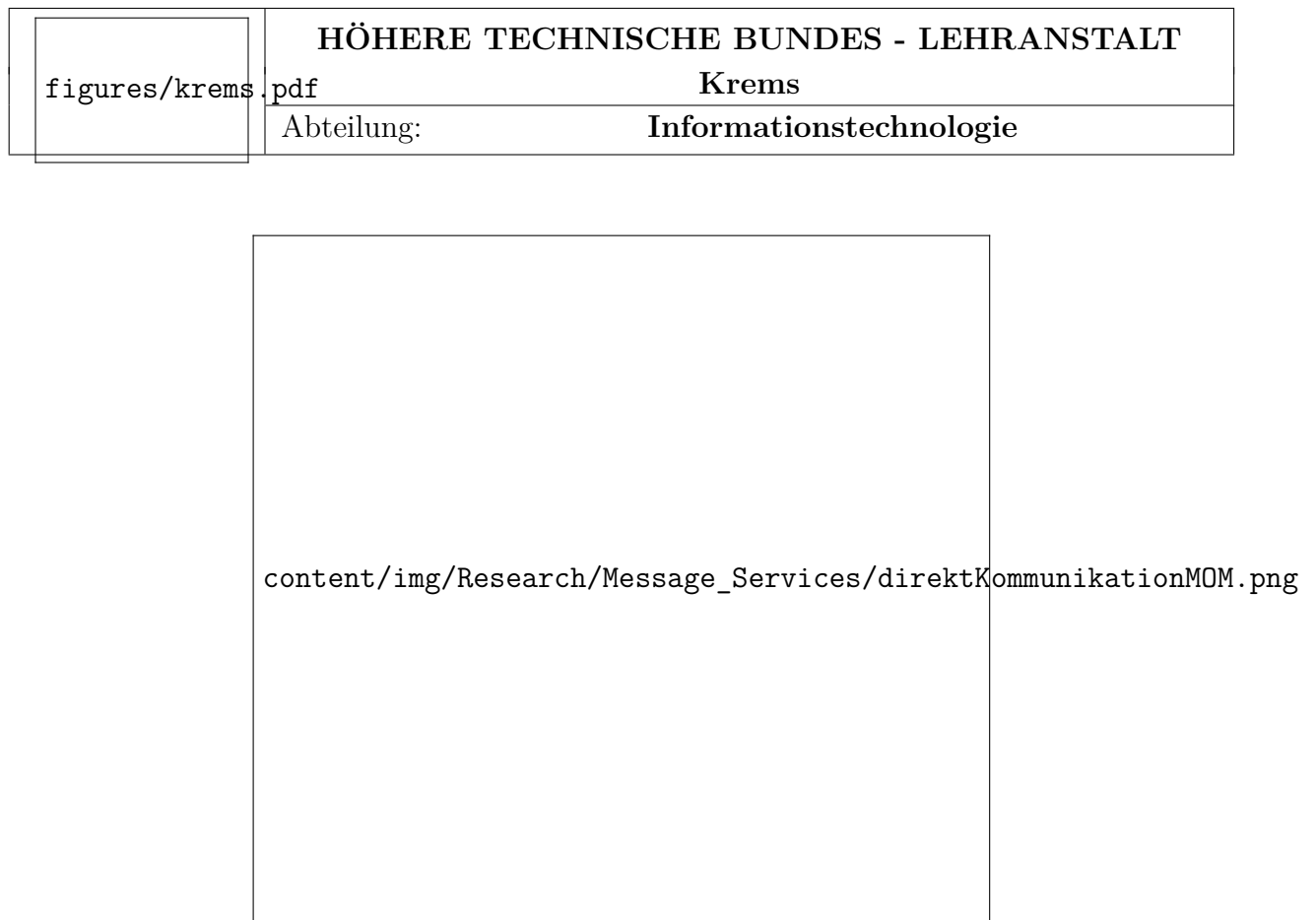


Abbildung 3.2.: Ein Beispiel für ein verteiltes System mit direkter Kommunikation
[3]

3.1.2.2. Verwendung einer gemeinsamen Datenbank

Eine weitere Möglichkeit des Informationsaustauschs könnte die Kommunikation über eine gemeinsame Datenbank sein. In dieser Architektur sind alle *Services* auf das gleiche Datenbank-Schema abhängig, welches die Koppelung reduziert. Durch die Zentralisierung der Kommunikationswege, wie es in der Abbildung 3.3 erkennbar ist, wird ein sehr übersichtlicher Aufbau des verteilten Systems geschaffen. Weitere *Services* können auch leicht integriert werden, da sie nur ein Datenbank-Schema entsprechend kommunizieren müssen. Wenn jedoch das Datenbank-Schema angepasst wird, müssen alle Module angepasst werden, dieser Prozess könnte mit großem Aufwand verbunden sein. Eine hohe Skalierbarkeit zu erreichen ist in den meisten Fällen sehr herausfordernd, da jedes Modul den zentralen Punkt belastet, insbesondere bei relationalen Datenbanken, da diese von Natur aus schwer zu skalieren sind. [3, 2]



Abbildung 3.3.: System mit einer Datenbank als zentrale Kommunikationskomponente

3.1.2.3. Nutzung von einem Messaging System

Bei der Verwendung eines *Messaging System* wird jegliche Kommunikation über einen zentralen Punkt geleitet, wie in der Abbildung 3.4 zu erkennen ist. Ein Sender schickt seine Nachricht zu dem Nachrichten-System, welches die Information einem oder mehreren Empfängern weiterleitet.

Durch den zentralisierten Aufbau bleibt das verteilte System übersichtlich und wartbar, was zu einer erheblich vereinfachten Fehlersuche führt. Die Kopplung der *Services* ist sehr niedrig, weil die Messaging-Systeme, eine Schicht zwischen den Empfänger und den Sender setzen, somit können auch beispielsweise alte *Services* leicht mit moderneren ersetzt werden.

Die Übertragung ist in diesem Fall vollständig asynchron, da die Kommunikationspartner nicht auf die Antwort des anderen warten müssen, weil die Nachricht dem Mittelsmann übergeben wird. Die Skalierbarkeit von verteilten Systeme, welche *Messaging Systeme* verwenden, sind durch diese asynchrone Natur grundsätzlich sehr hoch und können daher zu Spitzenzeiten auch die Last der Benutzer standhalten.

Zusätzlich wird auch die zuverlässige Nachrichtenübertragung sichergestellt. Zum Beispiel werden Nachrichten, die nicht zugestellt werden können, weil der Empfänger nicht erreichbar ist, zwischengespeichert, damit diese dem Empfänger zu einem späteren Zeitpunkt übertragen werden können.

Des Weiteren ermöglicht eine Architektur mit Messaging-Systemen eine einfache Erweiterbarkeit, denn durch das Hinzufügen eines weiteren *Services* werden keine bestehenden Kommunikationskanäle beeinträchtigt. Somit können neue Komponenten ohne großen Aufwand hinzugefügt werden, um die Funktionalität des Systems anzupassen oder zu erweitern.

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

Dies trägt dazu bei, dass das System flexibel bleibt und mit den Anforderungen des Unternehmens oder der Anwendung mitwachsen kann.

In Bezug auf die Latenz bietet die Verwendung von Messaging-Systemen oft niedrige Latenzzeiten, insbesondere durch die asynchrone Kommunikation. Dies ermöglicht eine effiziente und zeitnahe Kommunikation zwischen den verschiedenen Komponenten eines verteilten Systems.

[3, 2]



Abbildung 3.4.: Messaging System als zentraler Kommunikationsknoten [2]

3.2. Enterprise Messaging Systems

Messaging-Systeme bieten eine hervorragende Lösung für die Kommunikation in verteilten Systemen, wie es im Kapitel 3.1.2 dargelegt ist. In diesem Kapitel wird auf den Aufbau und die Funktionsweise von Nachrichten-Systemen eingegangen.

Durch ein *Messaging System* sind Sender (*Publisher*) und Empfänger (*Consumer*) stark entkoppelt, da sie über einen gemeinsamen Mittelsmann – dem Messaging-System – kommunizieren. Die Sender müssen Nachrichten dem *Messaging System* übertragen und die Empfänger erhalten die Nachrichten vom Nachrichten-System.

Damit die Nachrichten nicht durcheinander geraten, wird pro Anwendungsfall ein Kommunikationskanal (*Queue*) definiert, über welche sich die Kommunikationspartner austauschen können. Wie viele Sender oder Empfänger es in einem Kanal gibt, ist völlig beliebig. Beispielsweise könnte es in einem Monitoringsystem mehrere Sensoren geben, welche als *Publisher*

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT
	Krems
	Abteilung: Informationstechnologie

ihre gemessenen Daten einem Kanal übertragen. Des Weiteren könnte es noch einen *Consumer* geben, welche die Daten in eine Datenbank schreibt und einen anderen, welche die Daten für die Auswertung entgegennimmt. In der Abbildung 3.5 ist dieses Beispiel visualisiert.

Zusätzlich kann das *Messaging System* Nachrichten validieren, speichern, transformieren und durch vordefinierte Regeln den geeigneten Empfängern weiterleiten (*Routen*), dadurch wird ein einheitlicher und zuverlässiger Weg für die Kommunikation geboten. [2]



Abbildung 3.5.: Ein Beispiel für ein verteiltes System mit ein zentrales *Messaging System*

3.2.1. Enterprise Service Bus

Organisationen stehen oft vor der Herausforderung, verschiedene Technologien und Protokolle in das verteilte System integrieren zu müssen, um effektive Kommunikation und Interaktion zwischen ihren Systemen zu ermöglichen. In diesem Kontext spielen Messaging-Systeme eine entscheidende Rolle, da sie als Bindeglied zwischen den verschiedenen Komponenten dienen und den reibungslosen Austausch von Informationen unterstützen.

Die Verwendung verschiedener Protokolle wie REST, SOAP, JMX, AMQP oder RPC in einem verteilten System kann zu Komplexität führen, insbesondere wenn es darum geht, diese Protokolle miteinander zu integrieren. Hier kommt der *Enterprise Service Bus (ESB)* ins Spiel. Ein ESB fungiert als zentrale Kompatibilitätsschicht, die es ermöglicht, verschiedene Systeme und Protokolle nahtlos miteinander zu verbinden.

Diese Integration kann durch verschiedener Adapter realisiert werden, die Protokolle auf andere konvertieren, dadurch kann der ESB die Interoperabilität zwischen Systemen erleichtern. Dies ermöglicht eine effiziente Integration von bestehenden Systemen sowie die Anbindung von Systemen, die an bestimmte Protokolle gebunden sind. Trotz seiner Vorteile erfordert die Implementierung und Wartung eines ESB jedoch erheblichen Aufwand, um den unterschiedlichen Kommunikationsanforderungen der Protokolle gerecht zu werden. In der Abbildung 3.6 ist ein einfaches Anwendungsbeispiel eines ESBs dargestellt. [2, 4]

figures/krems	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT
	Krems
	Abteilung: Informationstechnologie



Abbildung 3.6.: Ein einfacher *Enterprise Service Bus*

3.2.2. Nachrichten

Das Konzept der Nachricht bildet das Herzstück der Kommunikation innerhalb eines Messaging-Systems. Eine Nachricht besteht in der Regel aus verschiedenen Bestandteilen, die ihre Struktur und Funktionalität definieren. Typischerweise umfasst eine Nachricht die folgenden Elemente [2]:

- **Header** – Der *Header* einer Nachricht enthält wichtige Metadaten, die für die Verarbeitung und Weiterleitung der Nachricht entscheidend sind. Dazu gehören Informationen wie das Encoding der Nachricht, Routing-Informationen zur Bestimmung des Zielorts der Nachricht und möglicherweise auch sicherheitsrelevante Daten, die für die Authentifizierung und Autorisierung relevant sind.
- **Body** – Der *Body* einer Nachricht enthält die eigentlichen Daten, die übermittelt werden sollen. Dies können beispielsweise Textnachrichten, strukturierte Daten im JSON- oder XML-Format, Binärdaten oder andere Formate sein, je nach den Anforderungen und dem Zweck der Nachricht. Jedoch sollte vermerkt werden, dass die Menge der Daten im Body gering gehalten werden sollte, damit verhindert wird, dass das *Messaging System* überladen wird. Wenn Daten von mehreren Gigabyte verschickt werden müssen, wäre die Übertragung eines Links für den Zugriff auf ein externes Speichersystem, wo die Daten sich befinden, die optimale Lösung.

Die Kombination aus Header und Body ermöglicht es, Informationen effizient und strukturiert innerhalb des Messaging-Systems zu transportieren, dadurch wird eine zuverlässige und effiziente Kommunikation innerhalb des Messaging-Systems gewährleistet.

3.2.3. Queues

Die *Message Queue* ist ein essenzieller Teil des *Messaging Systems*. Sie bieten die Funktionalität, Nachrichten für einen bestimmten Anwendungsfall im *Messaging System* zu speichern. Die Sender reihen Nachrichten in die *Queue* ein und die Empfänger nehmen die Nachrichten über die *Queue* entgegen, wie es in der Darstellung 3.7 zu erkennen ist.

Die standardmäßigen *Queues* folgen dem *First-In, First-Out (FIFO)* Prinzip. Das bedeutet, dass diejenigen Nachrichten zuerst bearbeitet werden, die als erste in die Queue eingefügt

wurden. Dies gewährleistet eine vorhersehbare Abwicklung der Nachrichten, wodurch eine ordnungsgemäße Kommunikation zwischen den verschiedenen Komponenten des Systems ermöglicht wird.

Die Sender und Empfänger können dabei völlig unabhängig voneinander mit der *Queue* interagieren. Dies bedeutet, dass Sender Nachrichten in die Queue einstellen können, ohne sich um die Verfügbarkeit der Empfänger kümmern zu müssen. Gleichzeitig können Empfänger Nachrichten aus der Queue abrufen und verarbeiten, ohne auf direkte Interaktion mit den Sendern angewiesen zu sein



Abbildung 3.7.: Eine *Queue* [3]

Es können viele Parameter von einer *Queue* konfiguriert werden, hierzu zählen unter anderen:

- Der Name der *Queue*
- Die Größe der *Queue*
- Der Sortieralgorithmus für die Nachrichten
- Der Schwellenwert für die Sicherung von Nachrichten

Eine spezielle Art der *Queue* sind die *Dead-Message Queues*, sie beinhalten Nachrichten, welche nicht zugestellt werden konnten. Typischerweise beinhalten diese Queues Nachrichten, die im *Body* kein gültiges Format oder eine ungültige Ziel-Queue haben. Der Zweck einer *Dead-Message Queue* besteht darin, Nachrichten aufzubewahren, die nicht erfolgreich verarbeitet werden konnten, damit sie später untersucht und möglicherweise erneut bearbeitet werden können. Durch die Verwendung einer *Dead-Message Queue* können Messaging-Systeme eine bessere Fehlerbehandlung und -toleranz bieten, da sie es ermöglicht, fehlgeschlagene Nachrichten zu isolieren und separat zu behandeln. [3]

3.2.4. Message Broker und Cluster

Die Kernkomponente eines Messaging Systems ist der *Message Broker*, dieser erhält die Nachrichten der *Services* und leitet sie anschließen zu den *Ziel-Services* weiter. Der *Broker* übernimmt unter anderen folgende Aufgaben:

- **Erhalt und Zustellen der Nachrichten** – Der Broker empfängt eingehende Nachrichten von den Services und sorgt für ihre zuverlässige Weiterleitung an die Ziel-Services.
- **Speichern der Nachrichten** – Der Broker kann Nachrichten vorübergehend speichern, um sicherzustellen, dass sie nicht verloren gehen, falls ein Ziel-Service vorübergehend nicht verfügbar ist.

- **Routing von Nachrichten** – Der Broker leitet eingehende Nachrichten an die entsprechenden Ziel-Services weiter, basierend auf definierten Routing-Regeln oder -Kriterien.

Um Ausfallsicherheit und Skalierbarkeit zu gewährleisten, können *Messaging Systems* in *Clustern* betrieben werden, die aus mehreren *Message-Brooklyn* bestehen. Diese Cluster ermöglichen die Replikation und Verteilung von Nachrichten über mehrere Broker hinweg. Es gibt zwei gängige Architekturen für Messaging-Cluster:

- **Master-Slave** – In dieser Architektur sind die Broker in *Master* und *Slave* unterteilt. Der Master-Broker übernimmt die Hauptaufgaben der Nachrichtenverarbeitung und -verteilung, während die Slave-Broker als Backup dienen und im Falle eines Ausfalls des Masters einspringen können.
- **Peer-to-Peer** – Hier haben alle *Broker* im *Cluster* denselben Status und jede Nachricht wird in mehreren Brokern repliziert und gespeichert. Dadurch wird eine höhere Redundanz und Ausfallsicherheit erreicht, da jede Nachricht mehrere Kopien in verschiedenen Brokern hat.

3.2.5. Messaging Models

Ein *Messaging System* verwendet eines oder mehrere Kommunikationsmodelle, welche die Grundlage für die Kommunikation der zu verbindenden Systeme definiert. [3]

3.2.5.1. Point-to-Point

In einer *Point-to-Point* Kommunikationsmodell gibt es nur einen Sender und einen Empfänger pro Nachricht. Im Fall, dass es mehrere potenzielle Empfänger für eine bestimmte Nachricht gibt, kann nur ein einziger sie tatsächlich erhalten. In diesem Szenario werden die Empfänger manchmal als *Competing Consumers* bezeichnet, da sie um den Empfang der Nachricht konkurrieren. Dieses Modell ist in der Abbildung 3.8 dargestellt. [3]

Dieses Modell bietet einige nützliche Anwendungsfälle. Zum Beispiel kann es für eine einfache Form des *Load Balancing* verwendet werden, indem die Nachrichten auf mehrere potenzielle Empfänger verteilt werden. Auf diese Weise können Arbeitslasten gleichmäßig auf mehrere Systeme oder Prozesse verteilt werden, um die Gesamtleistung und Effizienz zu verbessern. [3] Ein Real-World Szenario könnte wie folgt aussehen:

Die Prüfung auf Viren in Dateien sollte über das *Messaging System* auf mehrere Server verteilt werden. Dazu wird der Link zu der Datei von einem *Producer* in eine spezielle *Queue* geschickt. Eine Gruppe von Scan-Servern steht bereit, um die eingehenden Links zu den Dateien zu empfangen und sie auf Viren zu überprüfen. Der *Broker* wird diese Nachricht **einem** dieser Server weiterleiten und somit den Scan-Vorgang starten.



Abbildung 3.8.: Das *Point-to-Point Messaging Model* [3]

3.2.5.2. Publish-Subscribe

Im *Publish-Subscribe* Modell werden Nachrichten mittels Nachrichten-Thema (*Topics*) gruppiert, beispielsweise könnte man für jeden Anwendungsfall ein *Topic* erstellen. *Services*, welche Nachrichten zu einem bestimmten Thema erhalten möchten, melden sich für den Erhalt der Nachrichten an (*subscribe*). Wenn Sender die zu übertragende Information an *Topics* senden (*publish*), werden immer alle *Subscriber* des Themas die Nachricht erhalten, dies kann auch in der Abbildung 3.9 erkannt werden. Außerdem gibt es keine Einschränkungen hinsichtlich der Rollen, die ein *Service* einnehmen kann, das heißt ein *Service* kann gleichzeitig *Subscriber* und *Publisher* sein.

Ein anschauliches Beispiel wäre ein User-Activity Tracking-System, welche aus einem *Publisher* und zwei *Subscriber* besteht. Der *Producer*, welche die Aktivität des Benutzers sammelt, verschickt die Informationen zu einem bestimmten *Topic*. Die zwei Empfänger des *Topics* verarbeiten diese Nachrichten, wobei der eine die Aktivitäten in eine Datenbank speichert und der andere diese für ein Empfehlungssystem auswertet. [3]



Abbildung 3.9.: Das *Publish-Subscribe Messaging Model* [3]

3.2.6. Consumption Mode

Dieser Aspekt der Kommunikation trifft nur auf die Empfänger zu und beschreibt, wie die Empfänger die Daten erhalten. Im Wesentlichen gibt es in diesem Zusammenhang zwei verschiedene *Consum Modes*, welche in dem folgenden Kapitel erklärt werden. Die Wahl des richtigen Modus ist entsprechend den Anforderungen, die das verteilte System erfüllen sollte, zu treffen.

3.2.6.1. Push

In diesem Modus verteilt das *Messaging System* selber die Nachrichten an die Empfänger, das heißt immer, wenn eine neue Nachricht beim *Broker* ankommt, wird diese direkt den Empfängern weitergeleitet. Dadurch hat dieser Modus eine sehr gute Echtzeit-Performance, da die zu übertragende Information ohne Verzögerung an die Empfänger weitergeleitet wird.

Im *Push-Modus* besteht das Problem, dass der *Message-Broker* nicht über die Information verfügt, ob der Empfänger überhaupt in der Lage ist, weitere Nachrichten anzunehmen. Dadurch könnte der Empfänger überlastet werden, wenn in kurzer Zeit sehr viele Nachrichten übertragen werden, weil der Empfänger selbst nicht steuern kann, wie viele Nachrichten ihm zugeschickt werden. Um diese Problematik zu lösen, benötigt man ein *Flow-Control Mechanismus*, damit die Übertragungsrate geregelt werden kann. [3, 5]

3.2.6.2. Pull

Im *Pull-Modus* muss der Nachrichtenempfänger immer beim *Messaging System* nach neuen Nachrichten fragen. Auf diese Weise kann der Empfänger die Verarbeitung von Nachrichten in Übereinstimmung mit seiner eigenen Kapazität steuern. Jedoch muss der Empfänger in einem geeigneten Zeitintervall diese Abfragen durchführen, um eine hohe Latenz zu vermeiden. Wenn aber dieses Intervall zu niedrig gewählt worden ist, werden die *Message Broker* durch diese vielen Anfragen sehr stark belastet. Darüber hinaus ermöglichen diese Abfragen die Implementierung einer unkomplizierten Form von Lastenausgleich (*Load Balancing*), da jeder Empfänger eine Nachricht nur dann entgegennimmt, wenn er über die entsprechende Kapazität verfügt. [3, 5]

3.2.7. Quality-of-Service Garantien

Damit die Operation eines verteilten Systems zuverlässig funktioniert, muss ein *Messaging System* bestimmte Garantien zusichern. Jedoch haben gewisse Garantien einen negativen Effekt auf die Performance des Systems, beispielsweise wird für die Ordnung der Nachrichten in einem *Cluster-System* viel Rechenleistung benötigt. [5]

3.2.7.1. Zustellungsgarantien

Um eine zuverlässige Übertragung zwischen Consumer und Producer zu gewährleisten, sind verschiedene Strategien für die Nachrichtenübertragung verfügbar:

- **At-Most-Once** – Hier wird eine Nachricht höchstens einmal übertragen. Es besteht jedoch die Möglichkeit, dass die Nachricht während des Übertragungsvorgangs verloren geht.
- **At-Least-Once** – Diese Methode stellt sicher, dass eine Nachricht mindestens einmal vollständig übertragen wird. Es kann jedoch vorkommen, dass eine Nachricht mehrmals übertragen wird, um sicherzustellen, dass sie erfolgreich empfangen wurde.
- **Exactly-Once** – Bei dieser Strategie wird garantiert, dass jede Nachricht genau einmal übertragen wird. Dadurch wird sichergestellt, dass Duplikate vermieden werden und keine Nachrichten verloren gehen.

Die meisten *Messaging Systems* bieten die *At-Most-Once* oder *At-Least-Once* Garantie. Die *Exactly-Once* wird durch ihre Komplexität nur von manchen *Message Systeme* angeboten. [5]

Eine weitere Garantie, welche durch *Messaging Systems* erreicht werden kann, ist die *disk-based Retention*. Also die Speicherung der Nachrichten auf der Festplatte, welches einen großen Vorteil mit sich bringt, weil Nachrichten über einen längeren Zeitraum im *Messaging System* aufbewahrt werden können. Da die *Messages* auf der Festplatte persistiert werden und sich nicht nur im RAM befinden, kann kein Datenverlust auftreten. Es kann zum Beispiel während Wartungsarbeiten an den Empfängern trotzdem sichergestellt werden, dass keine Nachrichten verloren gehen. [6]

3.2.7.2. Ordnungsgarantien

Ein weiter wichtige Aspekt ist die Ordnung der Nachrichten, jedoch hat diese Garantie nur bei *Cluster-Systemen* Relevanz. Denn in einem *Cluster* ist die Synchronisierung der Ordnung über mehrere *Broker* hinweg sehr ressourcenintensiv. Die folgenden Strategien für die Nachrichtenordnung sind üblich [5]:

- **No-Ordering** – Bei dieser Strategie gibt es keine festgelegte Ordnung für die Nachrichten. Dies ermöglicht eine maximale Performance, da keine Ressourcen für die Synchronisierung der Nachrichten verwendet werden
- **Partition-Ordering** - Nachrichten werden innerhalb eines einzelnen Brokers geordnet, aber es gibt keine Garantie für die Ordnung über mehrere Broker hinweg. Diese Strategie bietet eine gewisse Ordnung, ohne die Performance übermäßig zu beeinträchtigen.
- **Global-Ordering** - Hier werden die Nachrichten über den gesamten *Cluster* hinweg geordnet. Dies bietet maximale Konsistenz, aber es geht auf Kosten der Performance, da die Synchronisierung der Nachrichtenordnung zwischen den Brokern sehr aufwendig sein kann.

Wenn die Ordnungsgarantie *Global-Ordering* aus Performancegründen nicht angewendet werden kann, ist es erforderlich, die Kommunikation der Services so zu gestalten, dass die Reihenfolge der Nachrichten keine Bedeutung hat. Jedoch kann diese Anforderung in der Umsetzung erhebliche Probleme verursachen. Die Synchronisierung von Aktionen und die Gewährleistung der Konsistenz der Daten können erschwert werden, da keine garantierte Reihenfolge der Nachrichten vorhanden ist. Dies erfordert eine sorgfältige Planung und Implementierung, um sicherzustellen, dass die Services trotz fehlender globaler Ordnung effizient und zuverlässig funktionieren.

3.2.8. Protokoll für die Kommunikation mit dem Broker

Das Protokoll zwischen den *Producer/Consumer* und den *Message Broker* bildet die Grundlage des Informationsaustausches und hat eine sehr wichtige Bedeutung, da es die Funktionen des *Messaging Services* definiert und eingrenzt. Die Kommunikationsstandards kann man in zwei Gruppen einteilen, in die *open-source* Protokolle und die *closed-source* Protokolle. Folgende weit verbreitete Protokolle werden zu den *open-source* Protokollen gezählt: *AMQP*,

XMPP, *REST* und *STOMP*. Die *open-source Messaging Protocols* sind den *closed-source* Protokollen in dem Aspekt Kopplung überlegen, da sie durch die offene Standardisierung die Abhängigkeit zu einem bestimmten *Messaging System* aufheben. *Closed-source* Protokolle sind oft Produkte von Veränderungen an existierenden *open-source* Protokollen oder völlig neue geschaffene Kommunikationsdefinitionen, welche nur für bestimmte Systeme gedacht sind. [5]

3.2.9. Latenz und Durchsatz

Der Begriff Latenz beschreibt die Zeit, die es benötigt, Nachrichten zwischen zwei Endpunkten über das *Messaging System* zu übertragen. Die folgenden Punkte haben den größten Einfluss auf die Übertragungszeit:

- Die Dauer für die *Metadaten-Verarbeitung* eines Pakets, wie zum Beispiel die Validierung oder das *Routing*.
- Die Zeit, welche in Anspruch genommen wird, für die *Replikation* eines Pakets. Dieser Aspekt trifft insbesondere auf *Messaging Systeme*, welche in einer *Cluster*-Architektur arbeiten, zu.
- Die Speicherzugriffsverzögerung, welche durch die Methode und Ort des Zugriffs bestimmt ist. Zum Beispiel sind der Zugriffe auf *RAM*-gespeicherte Daten weniger Zeit intensiv als Datenzugriffe auf eine Festplatte.
- Weitere Verzögerungen werden durch die Einhaltung der *Quality-of-Service* Garantien erzwungen, wie beispielsweise die Ordnungsgarantien, welche einen besonders negativen Effekt haben.

Low Latency Messaging Services minimieren diese Aspekte und können dadurch eine annähernde Echtzeitübertragung ermöglichen.

Der Durchsatz steht für die Menge an Daten, welche pro Zeiteinheit über das *Messaging System* übermittelt werden können. Damit die höchsten Durchsatzraten erreicht werden können, müssen die oben angeführten Punkte natürlich minimiert werden, aber es gibt noch einen weiteren Lösungsansatz: *Batch-Processing*. In dieser Methode werden mehrere Nachrichten gesammelt und auf einen Schlag übertragen, damit wird ermöglicht den *Overhead* zu reduzieren. Jedoch ist diese Technik bekannt für den Kompromiss zwischen Durchsatz und Latenz. Denn um mehrere Nachrichten auf einmal übertragen zu können, muss gewartet werden, bis entsprechend viele Nachrichten angekommen sind. Deswegen kann man bei manchen *Messaging Systemen*, wie *Kafka* [6], je nach Anwendungsfall bestimmen, wie viele Nachrichten akkumuliert werden. [5]

3.3. Vergleich von populären Enterprise Messaging Services

Zwei der populärsten Messaging-Systeme, die in der Softwareentwicklung weit verbreitet sind, sind *Apache Kafka* und *RabbitMQ*. Beide Systeme bieten leistungsstarke Funktionen für das Messaging und adressieren unterschiedliche Anwendungsfälle und Anforderungen. In diesem Kapitel wird ein detaillierter Vergleich zwischen diesen *Messaging Systeme* durchgeführt. Zusätzlich wird ihre Architektur, Funktionalitäten, Leistungsmerkmale, Einsatzszenarien und

weitere relevante Aspekte analysiert, um Entwicklern und Architekten dabei zu helfen, die richtige Entscheidung für ihr Messaging-System zu treffen. Dabei werden sowohl die Stärken als auch die Schwächen dieser beiden Systeme beleuchtet, um einen umfassenden Einblick in ihre Unterschiede und Gemeinsamkeiten zu bieten.

3.3.1. Apache Kafka

Kafka ist ein hoch skalierbares und verteiltes *Messaging System*, welches bekannt für die Fähigkeit große Datendurchsatzraten erreichen zu können ist. Dieses System wird in der Scala Programmiersprache geschrieben und hat eine sehr aktive Community, die das *open-source* Projekt ständig um Funktionen erweitert. Apache Kafka wird auch als *Streaming-Plattform* bezeichnet, da es die Anforderungen an *Stream-Processing* genügen kann. *Stream-Processing* ist die Datenverarbeitung an einen Fluss von Daten, also eine unendliche Serie an Daten. Das *Messaging System* von Apache ist für diesen Anwendungsfall gut geeignet, da es in der Lage ist, die benötigten Durchsatzraten zu erreichen und eine Reihe an *Stream-Processing* Funktionalitäten besitzt, wie zum Beispiel *Kafka Streams* oder *KSQL*. Außerdem verfügt Apache Kafka über *Connectors*, die zur Einbindung von externen Datenquellen oder Datensenden gedacht sind. Beispielsweise könnte man Daten über Kafka in einer Datenbank speichern oder über den *MirrorMaker* zwei Kafka *Cluster* verbinden. [7]

Folgende Auflistung zählt die wichtigsten Vorteile von Apache Kafka auf:

- **Hohe Skalierbarkeit** - Die Nachrichten werden in mehrere Partitionen aufgeteilt, die parallel beschrieben und ausgelesen werden können, was die Skalierbarkeit erhöht.
- **Hohe Verfügbarkeit** - Durch Replikation wird sichergestellt, dass *Messages* im *Cluster* redundant gespeichert werden, um Ausfallzeiten zu minimieren und die Verfügbarkeit zu maximieren.
- **Hohe Durchsatzraten** - Durch Batch-Verarbeitung werden große Mengen an Nachrichten gleichzeitig, effizient verarbeitet, was zu hohen Durchsatzraten führt.
- **Aufbewahrung der Nachrichten auf der Festplatte** - Die Nachrichten werden auf der Festplatte gespeichert, um Persistenz sicherzustellen und Datenverlust zu verhindern.

3.3.1.1. Architektur von Kafka

Grundsätzlich basiert die Architektur von Kafka auf das Messaging Model *Publish-Subscribe*. Die Nachrichten werden in verschiedene *Topics* unterteilt, und jedes Thema wird in mehreren Partitionen aufgeteilt. Diese Partitionen werden auf mehreren *Message Broker* verteilt und repliziert, da Apache Kafka normalerweise als *Cluster* arbeitet. Partitionen werden benötigt, um die Nachrichten gleichmäßig auf die *Message Broker* aufzuteilen und den Empfängern das gleichzeitige Auslesen eines Themas zu ermöglichen. Wie es in der Abbildung 3.10 zu sehen ist, verwendet Kafka die *Peer-to-Peer*-Replikationsmethode und Empfänger erhalten Nachrichten über den *Consumption Mode Pull*.



Abbildung 3.10.: Die Architektur des Apache Kafka Systems [5]

Die *Consumer* können sich zu Gruppen zusammenschließen (*Consumer-Group*) und können dadurch gemeinsam ein *Topic* verarbeiten. *Consumer Groups* können parallel mehrere Partitionen auslesen (siehe Abbildung 3.11), jedoch werden die *Messages* auf die Empfänger aufgeteilt. Dadurch bieten sich *Consumer-Groups* perfekt für Anwendung bei *Load Balancing* Use-Cases an. Außerdem benötigt Apache Kafka eine zusätzliche Applikation für das Verwalten und Koordinieren eines *Clusters*: *Zookeeper*, zum Beispiel wird diese Applikation zur fehlerfreien und parallelen Auslegung von *Topics* benötigt.



Abbildung 3.11.: Eine *Consumer-Group* verarbeitet gemeinsam ein Topic. [6]

Außerdem unterstützt Apache Kafka die Zustellungsgarantie *At-Least-Once* standardmäßig und die *At-Most-Once* Garantie kann bei Bedarf beim *Producer* eingestellt werden. Damit die *Exactly-Once* Zusicherung eingehalten werden kann, müssen aber Veränderungen am Ziel-Storage System bewerkstelligt werden.

Damit Kafka eine hohe Durchsatzrate erreicht werden kann, werden drei Methoden verwendet, die den *Overhead* minimieren:

- **Die Nutzung von *Zero-Copy-Methoden*:** Diese Technik wird verwendet, um Daten effizient zu übertragen, ohne dass Kopiervorgänge zwischen verschiedenen Pufferspeicherbereichen erforderlich sind. Im Kontext von Kafka wird diese Technik genutzt, um den *Overhead* bei der Datenübertragung zu minimieren und somit die Durchsatzraten zu verbessern.
- **Kafka überträgt Nachrichten in *Batches*:** Durch die Stapelung von Nachrichten und deren gemeinsame Übertragung wird die Anzahl der Netzwerkverbindungen minimiert und die Effizienz der Datenübertragung gesteigert, was zu höheren Durchsatzraten führt.
- **Nachrichtenkomprimierung:** Diese Methode komprimiert Nachrichten, bevor sie über das Netzwerk übertragen werden, was die benötigte Netzwerkbandbreite reduziert und die Übertragungsgeschwindigkeit erhöht, was zu einer insgesamt verbesserten Durchsatzrate führt. Insbesondere wird die Leistung gesteigert, bei der Übermittlung von Nachrichten, welche gut komprimierbare Datenformate verwenden, wie zum Beispiel *JSON*.

Zusammenfassend lässt sich sagen, dass Apache Kafka optimal für Anwendungsfälle geeignet ist, welche eine hohe Skalierbarkeit und hohe Ausfallsicherheit benötigen. Des Weiteren lässt sich Apache Kafka gut einsetzen in verteilte Systeme, welche die Anforderungen auf hohe Durchsatzraten bei der Übertragung von Nachrichten haben. Mit den Kafka *Connectors* ist es sogar möglich, die Anwendungsbereiche von einem *Enterprise Service Bus* zu decken. Jedoch ist bei der Verwendung von Kafka für Echtzeit-Kommunikation abzuraten, weil durch die Persistierung der Nachrichten auf der Festplatte Verzögerungen unvermeidbar sind.

3.3.2. RabbitMQ

RabbitMQ ist ein *open-source Messaging Service*, welcher um das Protokoll *AMQP* entwickelt wurde. Das System ist mit Erlang programmiert, da Erlang für das Entwickeln von verteilten Systemen geeignet ist, braucht es keine koordinierende Komponente wie *Zookeeper*. In der Abbildung 3.12 ist zu sehen, dass RabbitMQ auch als *Cluster* arbeiten kann, dafür werden die *Queues* auf mehrere *Broker* repliziert. Es gibt je *Queue* eine *Master-Queue* und eine oder mehrere *Backup-Queues*. Alle Operationen starten bei dem *Master* und werden zu den *Backup-Queues* weitergeleitet. Aber wenn die *Master-Queue* ausfällt, wird sofort eine *Backup-Queue* zur *Master-Queue* aufsteigen und somit ist eine hohe Ausfallsicherheit gegeben. Jedoch hat RabbitMQ kein gutes Design für Skalierbarkeit, weil die komplette Replikation der *Queues* nicht für hohe Auslastungen gut geeignet ist. *Exchanges* sind für das *Routen* der Nachrichten benötigt, sie können komplexe *Routing*-Anforderungen umsetzen, wie zum Beispiel die Verteilung der Nachrichten auf mehrere *Queues*.



Abbildung 3.12.: Die Architektur des RabbitMQ Systems [5]

Das RabbitMQ System ist ein *general-purpose Message Broker*, da es sehr viele Funktionalitäten implementiert, darunter befinden sich:

- **Push / Pull Consumption Mode** - RabbitMQ unterstützt sowohl den Push- als auch den Pull-Konsummodus, damit kann das *Messaging System* perfekt an den Anwendungsfall angepasst werden.
- **Unterstützung von standardisierte Protokolle** - RabbitMQ unterstützt eine Vielzahl von standardisierten Protokollen, darunter AMQP (Advanced Message Queuing Protocol), MQTT (Message Queuing Telemetry Transport), STOMP (Streaming Text

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

Oriented Messaging Protocol) und weitere. Diese Protokolle ermöglichen die Interoperabilität mit verschiedenen Clients und Systemen, was die Integration in bestehende Systeme erleichtert und die Flexibilität erhöht.

- **Point-to-Point und Publish-Subscribe Messaging Models** - RabbitMQ ermöglicht die Kommunikation mit dem Point-to-Point Modell oder mit dem Publish-Subscribe Modell
- **Disk Nodes oder Memory Nodes für RabbitMQ Cluster** - RabbitMQ bietet die Möglichkeit, Knoten im Cluster als Disk- oder Memory-Knoten zu konfigurieren. Disk-Knoten speichern Nachrichten dauerhaft auf der Festplatte, während Memory-Knoten Nachrichten nur im Arbeitsspeicher speichern.
- **At-Least-Once oder At-Most-Once Zustellungsgarantien** - Durch diese Zustellungsgarantien ist die Kommunikation über RabbitMQ zuverlässig, jedoch ist die *Exactly-Once* Garantie von RabbitMQ nicht unterstützt.

RabbitMQ bietet eine beeindruckende Bandbreite an Funktionalitäten, die seine Vielseitigkeit und Flexibilität unterstreichen. Diese Merkmale machen es zu einer attraktiven Option für die Integration in bestehende Systeme, da es sich nahtlos in unterschiedliche Architekturen einfügen kann. Trotz dieser Stärken steht RabbitMQ jedoch im Vergleich zu Apache Kafka in Bezug auf Skalierbarkeit und Performance zurück. Insbesondere hinsichtlich des Datendurchsatzes kann RabbitMQ nicht mit der Leistungsfähigkeit von Apache Kafka mithalten. Ein weiterer Unterschied besteht darin, dass RabbitMQ nicht die Möglichkeit bietet, *Exactly-Once* Semantik zu erreichen, was für bestimmte Anwendungsfälle von entscheidender Bedeutung sein kann. [5, 2]

4. Graphentheorie

Das Stromnetzwerk teilt aufgrund der starken Vernetzung die Eigenschaften von einigen Anwendungen von Graphvisualisierungen, wie zum Beispiel biologische Systeme, U-Bahn-Netzwerke und Dateisystemen. Aus diesem Grund werden die Grundlagen von Graphentheorie in diesem Kapitel erläutert.

4.1. Grundlagen eines Graphen

Graphen sind abstrakte mathematische Strukturen, die aus einer Menge von Knoten und den dazwischen verlaufenden Kanten bestehen. Jede Kante in einem Graphen repräsentiert eine Verbindung zwischen genau zwei Knoten. Die wissenschaftliche Disziplin, die sich mit der Untersuchung und Analyse solcher Graphen befasst, ist als Graphentheorie bekannt. In der Graphentheorie werden unterschiedliche Eigenschaften und Charakteristika von Graphen erforscht, wodurch sie als mächtiges Werkzeug in verschiedenen wissenschaftlichen, informatischen und ingenieurwissenschaftlichen Anwendungen dient. In diesem Kapitel werden verschiedene Eigenschaften von Graphen analysiert und mittels Abbildungen veranschaulicht. [8]

4.2. Ausrichtung

Die Besonderheit eines gerichteten Graphen ist die eindeutige Vorgabe der Kantenrichtung. Innerhalb dieses Graphen verbindet jede Kante präzise einen Anfangs- mit einem Endknoten und wird durch Pfeile repräsentiert, die eindeutig die Richtung der Beziehung zwischen diesen Knoten anzeigen. In der realen Welt könnten derartige gerichtete Verbindungen exemplarisch in einem sozialen Netzwerk und seinem *Following*-System auftreten, da die Möglichkeit besteht, dass eine Person A einer Person B folgt, jedoch nicht zwangsläufig umgekehrt. [8]

content/img/Research/Graphen/Ausrichtung.png

Abbildung 4.1.: Zwei Graphen dessen Unterschied die Ausrichtung ist

4.3. Konnektivität

Zusammenhängende Graphen haben die Eigenschaft keine Knoten aufzuweisen, welche vom Rest des Graphen isoliert sind. Diese Charakteristik impliziert das Fehlen isolierter Teil-

graphen innerhalb des Gesamtgefüges. Die Gewährleistung der Zusammenhangseigenschaft bedeutet, dass sämtliche Knoten durch Pfade miteinander verbunden sind, wodurch der Graph als kohärente und nicht in isolierte Teile zerlegbare Einheit betrachtet wird. Nicht zusammenhängende Graphen können an ihren isolierten Knoten erkannt werden. [9]

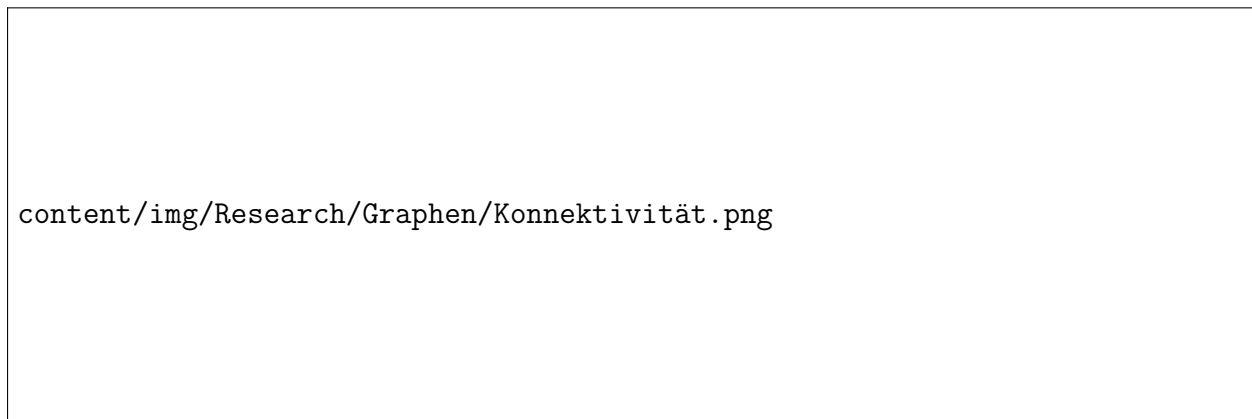


Abbildung 4.2.: Ein Graph all dessen Knoten verbunden sind (links); ein Graph, welcher nicht verbundene Knoten enthält (rechts)

4.4. Zyklen

Ein zyklischer Graph manifestiert sich durch die Existenz mindestens eines geschlossenen Pfades innerhalb seiner Struktur. Ein Pfad in diesem Sinne definiert sich als Abfolge von Kanten, dessen Anfangsknoten gleich dem Endknoten ist. Azyklisch wird ein Graph genannt, wenn seine Eigenschaften dem Gegenteil eines zyklischen Graphen entsprechen.

Die Ausführung von Algorithmen auf zyklischen Graphen erfordert besondere Achtsamkeit, da Zyklen potenziell zu komplexen, aufeinander abhängigen Situationen führen können. Das Ignorieren dieser Zyklen birgt das Risiko, dass solche Abhängigkeiten nicht aufgelöst werden. Daher ist eine präzise Analyse und Integration der zyklischen Strukturen in den algorithmischen Prozess unerlässlich, um die korrekte Verarbeitung von Daten und Abhängigkeiten zu gewährleisten. [8]

content/img/Research/Graphen/Zyklen.png

Abbildung 4.3.: Ein Graph, welcher keinen geschlossenen Kreis (Zyklus) enthält (links); ein Graph mit Zyklus (rechts)

4.5. Gewicht der Kanten

In einem gewichteten Graphen werden den Kanten zusätzliche numerische Werte zugeordnet, die als Gewichte bezeichnet werden. Diese Gewichte dienen dazu, verschiedene Arten von Beziehungen oder Kosten zwischen den Knoten zu repräsentieren. Die Gewichtung ermöglicht eine präzise Modellierung von unterschiedlichen Einflussstärken oder Ressourcenverbrauch entlang der graphentheoretischen Struktur, wie zum Beispiel Dauer und Länge einer Verbindungstrecke zweier Haltestellen bei einem öffentlichen Verkehrsnetzwerk. [8]

content/img/Research/Graphen/Gewicht.png

Abbildung 4.4.: Ein Graph, welcher gewichtete Kanten hat (links); ein Graph, dessen Kanten immer das gleiche Gewicht haben (rechts)

4.6. Vollständigkeit

Ein vollständiger Graph ist durch die Eigenschaft gekennzeichnet, dass jeder Knoten mit jedem anderen Knoten durch eine Kante verbunden ist. Diese charakteristische Vollständigkeit der Verbindungen impliziert eine maximale Interaktion zwischen den einzelnen Knoten des Graphen, wodurch sämtliche möglichen Kantenrelationen realisiert sind. Diese Einschrän-

kung der Flexibilität von den Eigenschaften eines Graphen kommen aus diesem Grund seltener vor, weswegen die meisten Graphen unvollständig sind. [8]

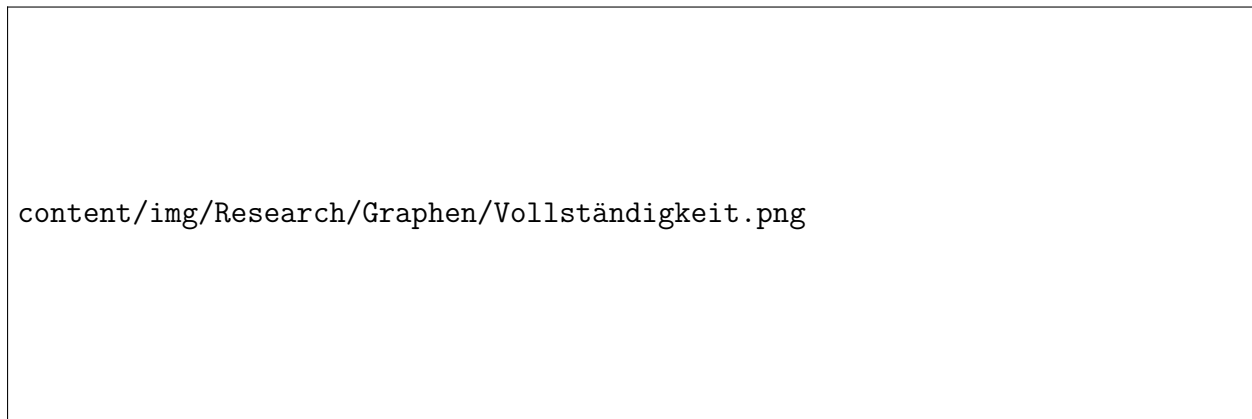


Abbildung 4.5.: Ein Graph, wo jeder Knoten mit allen anderen Knoten verbunden ist (links); nicht alle Knoten sind miteinander verbunden (rechts)

4.7. Bipartition

Die Besonderheit eines bipartiten Graphen ist die Unterteilung aller Knoten in zwei disjunkte Mengen, wobei innerhalb beider Mengen keine Kanten existieren. Knoten dürfen dementsprechend nur miteinander verbunden sein, falls sich diese nicht in derselben Menge befinden. Allen nicht bipartiten Graphen fehlt es an solch einer Eigenschaft. [8]

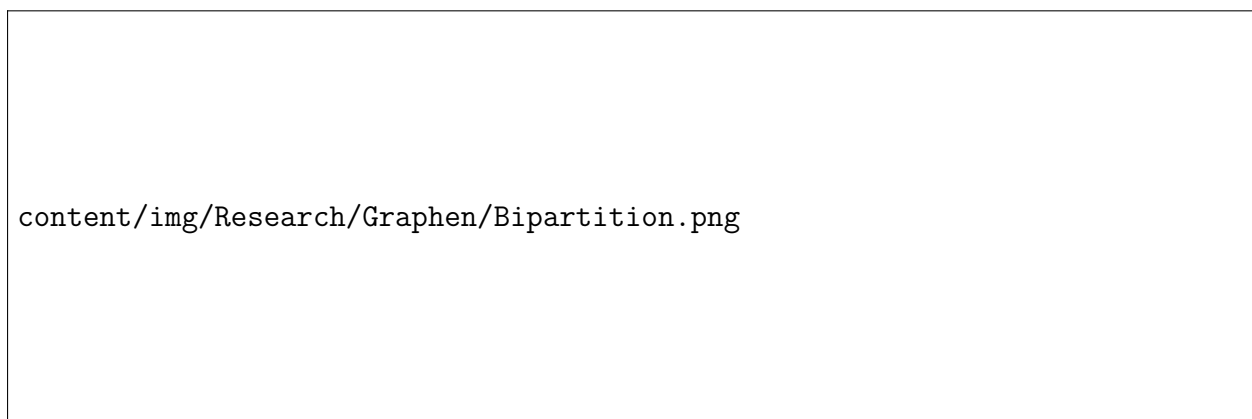


Abbildung 4.6.: Graph, welcher zwei Mengen ohne inhärente Verbindungen hat (links); keine Bipartition möglich beim rechten Graph

4.8. Planarität

Ein planarer Graph kann auf einer Ebene ohne Kantenkreuzungen dargestellt werden, was eine klare zweidimensionale Visualisierung ermöglicht. Im Gegensatz dazu erfordert ein nicht

planarer Graph Kantenkreuzungen bei der Darstellung auf einer Ebene, was die visuelle Erfassung erschwert. Die Planarität beeinflusst somit nicht nur die Struktur, sondern auch die graphische Repräsentation und Analyse des Graphen [10, 11].

content/img/Research/Graphen/Planarität.png

Abbildung 4.7.: Ein Graph ohne Schnittpunkte der Kanten, wenn alle Knoten verbunden sind (links); nicht möglich ohne Schnittpunkte (rechts)

4.9. Eulerscher Graph

Ein eulerscher Graph zeichnet sich durch die Existenz eines geschlossenen Pfades aus, der alle Kanten des Graphen genau einmal durchläuft. Diese charakteristische Eigenschaft ist als Eulerkreis bekannt und verleiht dem Graphen eine herausragende Struktur, da er eine systematische Durchquerung aller Verbindungen ermöglicht, ohne eine Kante zu wiederholen [12, 13].

content/img/Research/Graphen/EulerscherGraph.png

Abbildung 4.8.: Graph mit einem Zyklus, welcher alle Kanten einmal enthält (links); Graph ohne diesen Eulerkreis

4.10. Hamiltonscher Graph

Ein hamiltonscher Graph zeichnet sich durch die inhärente Eigenschaft aus, dass er die Existenz eines geschlossenen Pfades aufweist, welcher alle Knoten des Graphen exakt einmal durchläuft. Dieser geschlossene Pfad, bekannt als Hamiltonkreis, verankert die charakteristische Struktur des Graphen und ermöglicht eine vollständige, einmalige Durchquerung aller Knoten [12, 8].

Das Problem *Hamilton* beschreibt das Finden eines solchen Pfades (auch Kreis). Dieses Problem hat die Eigenschaft keinen effizienten Algorithmus zu besitzen, jedoch kann eine potenzielle Lösung effizient auf ihre Richtigkeit überprüft werden. Aufgrund dieser Eigenschaft gehört Hamilton zur Klasse NP. Spezifischer ist Hamilton sogar ein „NP-vollständiges“ Problem, eine Klasse mit den schwierigsten Problemen der Mathematik.

«Das Problem, einen Hamiltonschen Kreis in einem Graphen zu finden, bezeichnet man mit HAMILTON. Auch für dieses Problem ist kein effizienter Algorithmus bekannt.» - *Steffen Reith* [14]

content/img/Research/Graphen/HamiltonscherGraph.png

Abbildung 4.9.: Graph mit Zyklus, welcher alle Knoten beinhaltet (links); kein Hamiltonscher Kreis möglich (rechts)

Nachdem die Graphentheorie nun ausführlich erklärt wurde, widmen wir uns im nächsten Kapitel um die Visualisierung sowohl im Kontext der Benutzeroberfläche als auch im Kontext von stark vernetzten und komplexen Daten.

5. Visualisierung

Die Repräsentation von Objekten aus der realen Welt wird mittels Abstraktion und Filterung der essenziellen Merkmale deutlich sparsamer in Bezug auf Datenspeicherung. Die Visualisierung dieser abstrahierten Daten ist definiert als eine ansprechende Darstellung, um eine benutzerfreundliche Interaktion zu ermöglichen. Im Kontext der Effizienz ist die Aufgabe der Visualisierung eine nahtlose Verbindung zwischen der Datenspeicherung und dem Endbenutzer mit einer hohen Performance herzustellen. [15, 16]

In diesem Kapitel werden zuerst Eigenschaften der Benutzeroberfläche diskutiert, welche zu einer intuitiven Benutzererfahrung beitragen. Außerdem wird erläutert, von welcher Relevanz Interaktivität und Performance beim Implementieren einer Weboberfläche sind. Des Weiteren werden die unterschiedlichen Arten von Visualisierungsmethoden, zuerst bei einfachen und anschließend bei äußerst komplexen Daten, aufgelistet und erklärt.

5.1. Benutzerfreundlichkeit und Performance

Die Wirksamkeit einer Anwendung hängt nicht nur von deren Benutzerfreundlichkeit, sondern auch von deren Performance ab. Benutzerfreundlichkeit (auch *Usability*) ist die Leichtigkeit, mit den Daten interagieren zu können, und orientiert sich im Idealfall an den Denkweisen des Benutzers. Die Effektivität einer Applikation wird demnach stark von ihrer intuitiven Benutzerbarkeit beeinflusst. [17, 18]

Zunächst werden in diesem Kapitel die drei Begriffe Intuitivität, Interaktivität und Performance aus diesem Grund im Kontext des *UI-Designs* näher beschrieben. Außerdem werden grundlegende grafische Elemente des UI-Designs erwähnt und erklärt.

5.1.1. Intuitivität

Benutzer manifestieren die Erwartung, Applikationen ohne Konsultation von Gebrauchsanweisungen und Handbüchern bedienen zu können. Diese Eigenschaft einer Anwendung nennt sich Intuitivität und ist ein Grundsatz moderner Programmsysteme. Intuitivität minimiert die Lernmenge initiativ erheblich. [19]

Als Beispiel nehme ich nun die Filterung von Daten her. Benutzer wollen mithilfe der Filterung schneller an die Daten kommen. Aus diesem Grund sollte die Filterung nicht unnötig kompliziert sein, sodass Benutzer zuerst die Filterungsmöglichkeiten verstehen müssen, um diese richtig anwenden zu können. Als Designer dieser Filter muss man sich deswegen einige Fragen stellen: [20]

- Welche **Datentypen** sind in meinen Daten enthalten? Boolische Werte können viel intuitiver mittels Ein-Aus-Schalter, als mit Checkboxes, gefiltert werden.
- Welche Daten wird der Benutzer am **häufigsten filtern**? Der Status eines Paketes ist interessanter als der aktuelle Standort.
- Wo soll ich die Filtermöglichkeiten **platzieren**? Je nachdem, ob die Filter seitlich oder direkt bei dem Datenkomponenten angezeigt werden, kann man den Platz unterschiedlich nutzen und unterbewusst verdeutlichen, zu welchen Komponenten die Filter eigentlich gehören.

- Zu welchen **Zeitpunkten** sollen die Daten neu geladen werden? Während sich für langsame Webseiten ein „Anwenden“-Button über alle Filtermöglichkeiten eignet, können einfachere Filter direkt angewandt werden.

Es gibt natürlich noch weitere „best-practise“ Tipps, wie zum Beispiel bestimmte Voreinstellungen, welcher der Benutzer öfters brauchen kann. Wie die Filterung schlussendlich gestaltet wird, hängt ganz von den Daten ab. [20]

Ein weiterer wichtiger Bestandteil einer barrierefreien Benutzeroberfläche sind *Icons*, da diese Informationen visuell vermitteln und nicht an eine bestimmte Sprache gebunden sind. Somit müssen Benutzer die Systemsprache nicht zwingen verstehen, um eine Anwendung zu bedienen. Dies ist besonders bei Anwendungen für ein internationales Publikum von großer Bedeutung. Eine hohe Priorität hat deswegen die überlegte Auswahl eines verständlichen Iconsets. [21]

Ein weit verbreiteter Mythos des Navigationsdesigns ist die „Three-Click Rule“. Die Grundrichtlinie dieser Regel beschränkt die Anzahl an Klicks oder Eingaben jeglicher Art auf drei, um zu jedem Bereich im Programmsystem zu gelangen. Jedoch stellte sich bei detaillierteren Untersuchungen heraus, dass die Anzahl von Klicks alleine weder Einfluss auf die Benutzerzufriedenheit noch die Erfolgsrate der Applikation hat. Kritik äußert sich demnach aufgrund der Vernachlässigung der kontextuellen Komplexität und individuellen Benutzerziele. [22, 23]

Als nächstes wird die Interaktivität einer Webseite besprochen und welche Auswirkungen diese auf die Benutzererfahrung hat.

5.1.2. Interaktivität

Interaktivität beschreibt eine Eigenschaft eines Inhalts, den Benutzer dazu zu verleiten, sich mit dem Inhalt auseinander zu setzen. Auch wenn das ein wenig geschwollen klingt, trifft diese Beschreibung die Grundidee von Interaktivität recht gut. Der Benutzer ist nicht einfach ein Betrachter des Inhaltes, sondern viel mehr Teil des Prozesses. Es kann sich bei Interaktivität um einfache Hyperlinks, online Quizze, Videos oder sogar Virtual Reality handeln. Wie eine Studie zeigt, bevorzugen 45 Prozent der Menschen interaktive Inhalte. Dies ist unter anderem auch ein Grund, warum Menschen gerne Videospiele spielen, da die Interaktivität dort besonders hoch ist. [24]

«Interaktivität ist das Potenzial eines technischen Einzelmediums oder einer Kommunikationssituation, das interaktive Kommunikation begünstigt, also den Prozess der Interaktion.» - *Christoph Neuberger* [25]

Die Integration effektiver Filter- und Suchfunktionen in Datenvisualisierungssystemen spielt eine Schlüsselrolle bei der Optimierung der Benutzererfahrung und der gezielten Extraktion relevanter Informationen. Untersuchungen haben deutlich gemacht, dass die gezielte Implementierung von sorgfältig gestalteten Filteroptionen in Datenvisualisierungssystemen die Nutzerbefähigung zur gezielten Suche nach spezifischen Datenpunkten oder Mustern in hohem Maße verbessert. Dies resultiert wiederum in einer signifikanten Steigerung der Effizienz innerhalb des Datenmanagements. [26]

Eine wichtige Komponente ist die Anpassungsfähigkeit der Filter, um unterschiedlichen Anforderungen gerecht zu werden. Die Möglichkeit, Filterkriterien zu kombinieren oder anpass-

bare Parameter festzulegen, ermöglicht eine feinere Steuerung und eine präzisere Datenauswahl. Dies trägt dazu bei, dass Benutzer ihre Anfragen flexibel an die Komplexität des Datensatzes anpassen können. [27]

Forschungen von Shneiderman et al. betonen die Bedeutung von *Dynamic Queries*, einer Technik, bei der Benutzer unmittelbares Feedback zu ihren Filteranfragen erhalten. Diese Echtzeit-Rückmeldung erleichtert Benutzern eine explorative Analyse. Des Weiteren haben Studien gezeigt, dass die Integration von Vorschlägen während der Eingabe (Autovervollständigung) und die Verwendung von Synonymen die Benutzerfreundlichkeit der Suchfunktionen verbessern. Diese Erkenntnisse verdeutlichen die Relevanz einer kontinuierlichen Forschung und Weiterentwicklung von Filter- und Suchmechanismen, um den sich ständig ändernden Anforderungen der Benutzer gerecht zu werden. [28, 26]

5.1.3. Performance

Die Effizienz von Webanwendungen spielt eine entscheidende Rolle in der Gestaltung einer ansprechenden Benutzererfahrung. Zahlreiche Studien haben sich mit der Leistungsoptimierung von Webseiten befasst und zeigen, dass eine schnellere Ladezeit einen unmittelbaren Einfluss auf die Zufriedenheit und Interaktion der Nutzer hat. In diesem Zusammenhang präsentiert eine umfassende Untersuchung von Souders, wie die Minimierung von HTTP-Anfragen, das effiziente *Caching* von Ressourcen und die Reduzierung von DNS-Lookups als kritische Elemente für die Optimierung der Webseitenleistung gelten. [29]

Besonders relevant für die Verbesserung der Benutzerfreundlichkeit sind Optimierungstechniken wie *Lazy Loading* und *Indexing*. Lazy Loading ermöglicht es, Ressourcen nur dann zu laden, wenn sie vom Nutzer angefordert werden, was die Interaktivität beschleunigen kann. In Bezug darauf zeigen Forschungen auf, dass eine effiziente Indexierung von Inhalten die Navigation und den Zugriff auf relevante Informationen für die Nutzer erheblich verbessert. [30, 31, 32, 33]

Aktuelle Studien von Hogan zeigen, dass die Optimierung von HTML, CSS und Bilddateien sowie die Verbesserung der Performance unter Berücksichtigung der steigenden Prozentwerte der mobilen Internetnutzung von besonders hoher Wichtigkeit sind. Diese Erkenntnisse betonen die Bedeutung von Leistungsoptimierungstechniken in der Webentwicklung und vor allem auch in der mobilen Appentwicklung und verdeutlichen, wie diese direkte Auswirkungen auf die Benutzerfreundlichkeit haben können. Die ganzheitliche Berücksichtigung von Aspekten wie Minimierung von HTTP-Anfragen, effizientes Ressourcen-Caching, Lazy Loading und Indexing kann somit als Schlüssel zur Schaffung einer optimalen Benutzererfahrung dienen. [31, 34]

5.2. Arten von Datendarstellungen

Die Visualisierung von hochvernetzten Daten ist heutzutage von zentraler Bedeutung, um die Herausforderungen bei der umfassenden Analyse und Deutung dieser komplexen Datenstrukturen zu bewältigen. Datengefüge solcher Art, die sich beispielsweise in sozialen Netzwerken, biologischen Systemen, Nahrungsketten, Dateisystemen oder sogar in der Struktur

der Sprache manifestieren, erfordern spezialisierte Visualisierungstechniken zur Extraktion ihrer inhärenten Muster und Strukturen. [35]

Bevor die bewährtesten Darstellungen von stark vernetzten Daten erläutert werden, werden noch Visualisierungen von kleineren und einfacheren Datenmengen erklärt.

5.2.1. Darstellung von einfachen Daten in Tabellen, Diagrammen und Kennzahlen

Die Visualisierung von Daten definiert sich als eine grafische Repräsentation, welche korrelierende Daten und logische Relationen nicht nur kommuniziert, sondern auch zu besseren Entscheidungen führt, da diese sachlich unterlegt und begründet sind. Einfache Visualisierungen, wie Balken-, Säulen-, Linien-, Kreis- oder Punktwolkendiagramme, genauso wie Wärmekarten, Tabellen und Kennzahlen in Prozent und anderen Einheiten eignen sich wunderbar für Berichte aus Marketingkampagnen, Leistung eines Vertriebsteams, Produktakzeptanzraten und vor allem für moderne Dashboards. Dashboards geben dem Benutzer einen schnellen Überblick über die verschiedensten aktuellen Daten, damit der Benutzer priorisieren kann, welche Daten näher unter die Lupe genommen werden müssen, damit dies infolgedessen zu besseren Entscheidungen führt. [36]

In Abbildung 5.1 sieht man einige Beispiele dieser einfachen Darstellungsarten aufgezeichnet.



Abbildung 5.1.: Einfache Datenvisualisierungen in Form von Werten, Diagrammen, Tabellen und Karten [37]

Die Visualisierungen in Abbildung 5.1 halten sich auch an wichtige Empfehlungen, um eine erfolgreiche Darstellung zu erstellen. Grafiken sollen laut Katy French nämlich eine vollkommene Geschichte erzählen. Dabei wird vor einer Überflut an Daten innerhalb der Visualisie-

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

rung gewarnt, welche vom wesentlichen Grundgedanken der Darstellung ablenken würden. [38]

Ein weiterer Tipp bezieht sich auf die Auswahl einer bestimmten primären Farbe, welche mittels Helligkeit und Sättigung mehrere Farbtöne für das Diagramm erzeugt. Zusätzlich ist es immer sinnvoll, Beschriftungen gut leserlich zu gestalten, die Daten intuitiv zu ordnen und keine Elemente in Darstellungen einzufügen, welche zum Beispiel bei Präsentationen nur dazugesagt werden. [38]

Beim Designen eines Dashboards sind die Differenzen der verschiedenen Diagramme zu beachten. Denn nicht jedes Diagramm kann von Menschen mit der gleichen Effizienz interpretiert werden und einige Datensätze können in bestimmten Diagrammen nicht gut visualisiert werden. Balkendiagramme stellen Prozentwerte beispielsweise deutlich veranschaulicher dar, als Kreisdiagramme, da Kreisdiagramme die Bogenlänge das Maß für die Proportionalität. Nur durch richtige Anordnung der Kreissegmente können Benutzer die Daten richtig extrahieren. In Balkendiagrammen werden die Datensätze oft absteigend nach Proportionalität sortiert. [39]

Auch zwischen Säulen- und Balkendiagrammen gibt es Differenzen. Denn intuitiv befindet sich der Zeitfaktor in jedem Diagramm auf der x-Achse. Deswegen werden Vergleiche über einen gewissen Zeitraum immer mittels Säulendiagrammen dargestellt. Handelt es sich um einen längeren Zeitraum, dessen Gruppierungen die Anzahl 15 überschreiten, wird empfohlen, ein Liniendiagramm stattdessen zu wählen, da sonst zu viele Säulen vorhanden sind. Balkendiagramme eignen sich wiederum besonders für kategoriale Vergleiche, wie zum Beispiel verschiedene Genre bei Büchern und dessen Beliebtheit (Abbildung 5.2). Gruppierte Säulen- beziehungsweise Balkendiagramme sind immer dann zu wählen, wenn pro Zeiteinheit beziehungsweise Kategorie mehrere Datensätze bestehen. Falls diese Datensätze Mengen beschreiben, wie zum Beispiel Anzahl an Frauen und Männern - zusammengerechnet ergibt sich die Anzahl an Personen -, sind gestapelte Säulen- und Balkendiagramme eindeutig zu bevorzugen. [40]



Abbildung 5.2.: Beliebtheit der verschiedenen Buchgenre im Jahr 2016 [41]

Jede Diagrammart hat demnach ihre Daseinsberechtigung und soll auch für den spezifischen Anwendungsfall herangezogen werden. [40]

5.2.2. Darstellung von komplexen Daten in Graphen, Heatmaps und Diagrammen

Das Visualisieren von Daten allgemein - unabhängig von der Menge - unterstützt uns wesentlich bei Entscheidungen. Diese Hilfe bei der Findung relevanter Entscheidungen ist einer der essenziellsten Existenzgründe für Darstellungen im Allgemeinen. Die Herausforderung bei der grafischen Aufbereitung von stark vernetzten Daten liegt in der schieren Menge der Daten. Man spricht von *Big Data*. [36, 42]

Pohan Lin erklärt in seinem Artikel die Schwierigkeit, große Datenmengen auf einfachen Bildschirmen intuitiv zu visualisieren, sodass Muster und Zusammenhänge erkannt und dementsprechend auch objektiv sinnvolle Schlüsse aus Daten gezogen werden können. Ebenso erläutert Eberhard Heins in einem Artikel von 2017 die Herausforderungen welche mit der Visualisierung von Big Data kommen. Mit steigender Anzahl an Knoten und Kanten in einem Graphen wird dieser unübersichtlicher, da Muster in dem „Farbteppich“ nicht extrahiert werden können, was die Entscheidungshilfe signifikant reduziert. Ebenso leidet die Orientierung und Leserlichkeit der Darstellung darunter. Nur unter effizienter und übersichtlicher Umsetzung einer Big Data-Darstellung können die Vorteile der Visualisierung genutzt werden. [42, 43]

Um ein ungefähres Bild von der schieren Menge an Informationen bei hoch korrelierenden Daten zu bekommen, wird Abbildung 5.3 hier kurz erläutert. Es handelt sich um eine Darstel-

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT
	Krems
	Abteilung: Informationstechnologie

lung des sozialen Netzwerkes LinkedIn. In diesem Graphen hat jeder Konten die Bedeutung einer Person und jede Kante beschreibt eine Art von Beziehung zwischen den beiden anliegenden Personen. Die Extraktion hilfreicher Informationen kann bei Anbetracht dieser Grafik vergessen werden.



Abbildung 5.3.: Stark vernetzte Daten in einem Graphen (LinkedIn Netzwerk) [44]

Die Komplexität eines Graphen kann mittels unterschiedlichen Ansätzen reduziert werden. Logischerweise führt eine Reduktion der Datenmenge automatisch zu einer übersichtlicheren Visualisierung und infolgedessen besseren Orientierung und Leserlichkeit beziehungsweise Erkennbarkeit. Jedoch kommt die Reduktion mit dem großen Nachteil des Informationsverlustes. Ein weiterer Ansatz definiert aggregierte Visualisierungstechniken als Zusammenfassung gewisser Informationen (*Clustering*). Im Folgenden werden verschiedene Darstellungsmethoden im Detail analysiert. [43]

5.2.2.1. Vereinfachung der komplexen Daten mittels Clustering

Graph-Clustering ist eine Methode zur Vereinfachung eines komplexen Graphen, indem Gruppierungen, Muster, Zusammenhänge oder Strukturen mittels Algorithmen extrahiert werden. Diese neuen Daten sind entscheidend für Datenwissenschaftler. Denn aus den Erkenntnissen dieser können fundierte Entscheidungen in verschiedenen Bereichen, wie zum Beispiel Marketing, Bioinformatik oder auch die Entdeckung von Fehlern oder Sicherheitslücken innerhalb eines Netzwerkes, getroffen werden. [45]

Die Algorithmen hinter Graph-Clustering sind unterstützt von maschinellem Lernen. Die künstliche Intelligenz versucht hierbei Gemeinsamkeiten der einzelnen Nodes innerhalb des Graphes zu finden. Diese Ähnlichkeiten können entweder direkte Eigenschaften der Knoten

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT
	Krems
	Abteilung: Informationstechnologie

sein oder Berechnungen aufgrund der Konnektivität zwischen den Knoten im Graphen. In Abbildung 5.4 kann auf der rechten Seite die Gruppierung der Knoten des Ausgangsgraphen (links) mittels farblichen Markierungen festgestellt werden. [45]

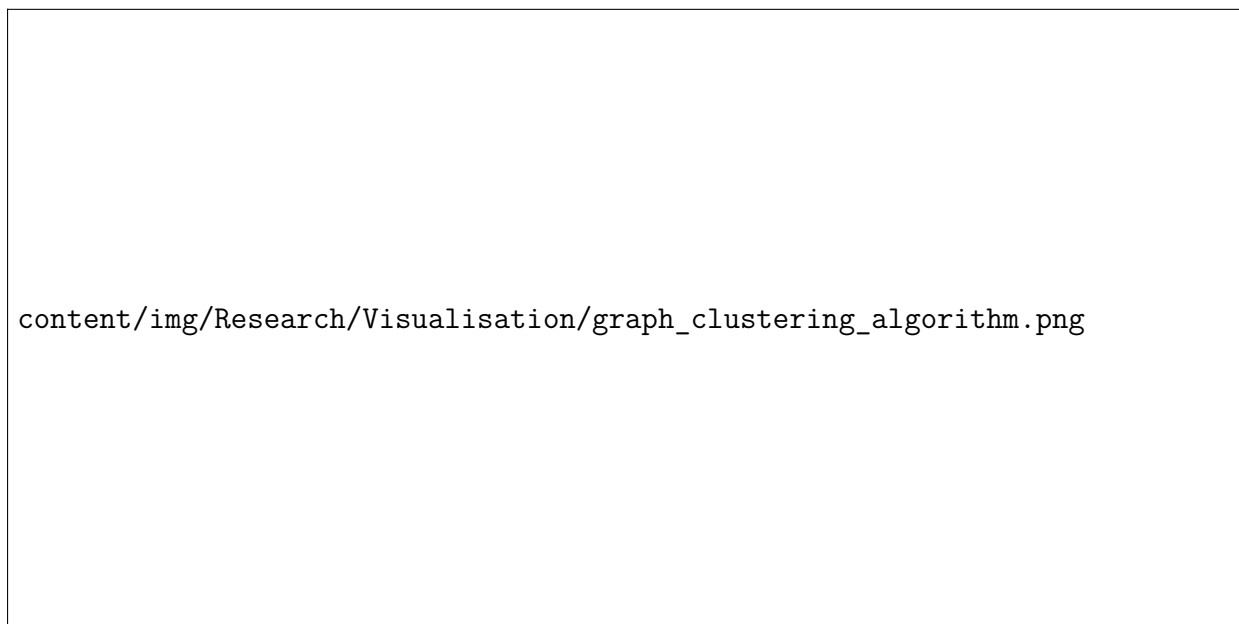


Abbildung 5.4.: Clustering eines Graphen [45]

Das Clustering eines Graphen kann auch nur auf irrelevante Daten angewandt werden, welche jedoch gruppiert in der Visualisierung beibehalten werden sollen. Beispielsweise können aktuelle Daten in der Mitte möglichst detailliert aufbereitet sein, während Daten aus vergangenen und demnach veralteten Jahre rundherum aggregiert dargestellt sind. [43]

5.2.2.2. Vereinfachung der verknüpften Daten mittels farblichen Abstufungen

Solange die Zugänglichkeit einer Grafik auch für Menschen mit Farbsehschwächen gegeben ist, sind Farben ein wunderbarer Weg, um verschiedene Daten in Diagrammen oder auch Grids zu visualisieren. Sogenannte *Heatmaps* eignen sich besonders gut, um eine große Datenmenge geordnet und gruppiert darzustellen. Dabei werden die Daten in einem Diagramm mit zwei Achsen farblich auf Rechtecken dargestellt, wobei die Farbe Auskunft über die Intensität in der jeweiligen Zeile und Spalte gibt. Empfohlen wird eine Verwendung einer Legende, um die Bedeutung der Farben in der Heatmap zu erklären. [46, 47, 48]

Beispielsweise kann man aus der Abbildung 5.5 deutlich ablesen, dass der Benutzer fast ausschließlich an Wochentagen arbeitet. Samstage und Sonntage sind nämlich in den meisten Fällen grau, was laut Legende bedeutet, dass keine Contributions von diesem Benutzer an diesen Tagen gemacht worden sind. Und genau solche Informationen sind in vielen anderen Bereichen entscheidend für Datenwissenschaftler.

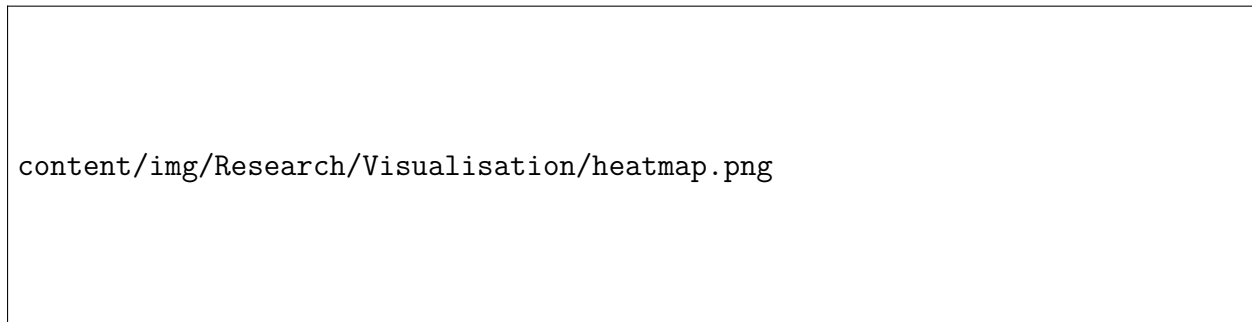


Abbildung 5.5.: Heatmap der Beiträge eines GitHub Benutzers im Jahr 2022 [46]

5.2.2.3. Vereinfachung der vernetzten Daten mittels Interaktivität

Die Visualisierung von Graphen inkludiert meistens eine Zoom- und Filterfunktionalität. Neben diesen Orientierungshilfen gibt es auch die Möglichkeit, Daten in Diagrammen und Graphen mittels Lupentechniken zu vereinfachen. Dabei werden gewisse Daten mittels Verzerrungen hervorgehoben und die restlichen Daten nur verkleinert am Rande angezeigt. Das explorative Verhalten der Visualisierung wird umso mehr verbessert, wenn die Lupenfunktionalität mit dem Benutzer interagiert. Eine besonders intuitive Interaktionsmöglichkeit ist die Lupenverfolgung des Mauszeigers. In anderen Worten kann der Benutzer die Position der Lupe in der Grafik mittels Maus direkt beeinflussen. [49]

In Abbildung 5.6 kann das Prinzip der Lupe auf einem Grid gut erkannt werden. Die Verzerrungsmethode heißt hierbei *Fisheye*.

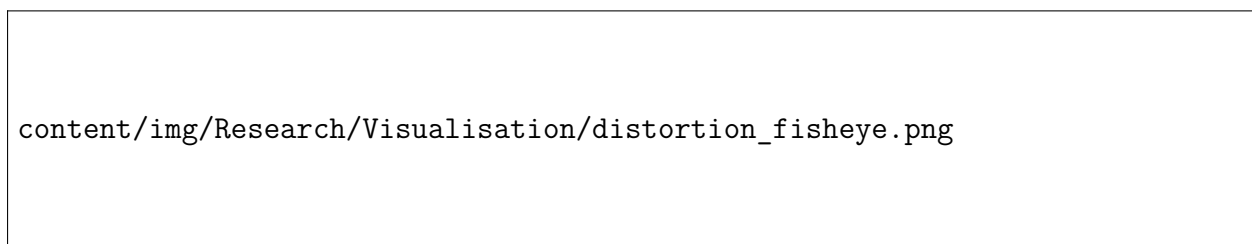


Abbildung 5.6.: Demonstration der Lupentechnik Fisheye auf einem Grid [49]

Auch, wenn ein statisches Bild den Effekt eines Fischauges in einem Graphen nicht perfekt veranschaulichen kann, sieht man in Abbildung 5.7 einen Graphen, dessen Knotenpunkt „Brujon“ momentan mittels Lupe analysiert wird.

content/img/Research/Visualisation/fisheye_graph.png

Abbildung 5.7.: Demonstration des Fischauges in einem Graphen [Source: GitHub Gist]

Die Lupentechnik muss nicht immer mit dem Fisheye-Effekt visualisiert sein. Es gibt zum Beispiel auch die Möglichkeit, eine *kartesische Verzerrung* (Abbildung 5.8) interaktiv zu verwenden. Dabei werden die Skalierungen der Achsen je nach Mausposition so angepasst, dass es sich so anfühlt, als wäre der Inhalt darunter am nächsten beziehungsweise größten.

content/img/Research/Visualisation/distortion_cartesian.png

Abbildung 5.8.: Demonstration der Lupentechnik Kartesisch auf einem Grid [49]

5.2.2.4. Vereinfachung der vielschichtigen Daten mittels Reduktion

Die wohl einfachste Methode, stark vernetzte Daten intuitiv und verständlich darzustellen, ist das Entfernen gewisser Datensätze beziehungsweise gewisse Attribute oder Dimensionen von Datensätzen aus der Grafik durch spezifische Filterung der Daten. Dadurch werden nur die wichtigsten Daten angezeigt und der Benutzer kann diese interpretieren und somit vernünftige Entscheidungen daraus schließen. Die Reduktion der Daten hat jedoch den Nachteil, dass einige, eventuell essenzielle Informationen verloren gehen. Bestimmte Zusammenhänge und Strukturen der Daten können nämlich nicht extrahiert werden, wie es jedoch bei anderen Visualisierungsmöglichkeiten (wie zum Beispiel Clustering) der Fall ist. Dieser unvermeidliche Informationsverlust sollte bei der Reduktion minimiert werden, um in Bezug auf Richtigkeit möglichst nahe an den ungefilterten Daten zu bleiben. [43, 50]

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT
	Krems
	Abteilung: Informationstechnologie

5.2.2.5. Darstellung der vereinfachten Daten durch Mischformen

„Best practice“-Visualisierungen vermischen verschiedene Formen der Vereinfachung und erstellen Grafiken, welche die wesentlichsten Informationen kurz und knapp auf den Punkt darstellen. Beispielsweise hat Google in der Dokumentation von Google Maps eine geobasierte Heatmap¹ herangezogen, welche in Abbildung 5.9 zu Demonstrationszwecken abgebildet ist.

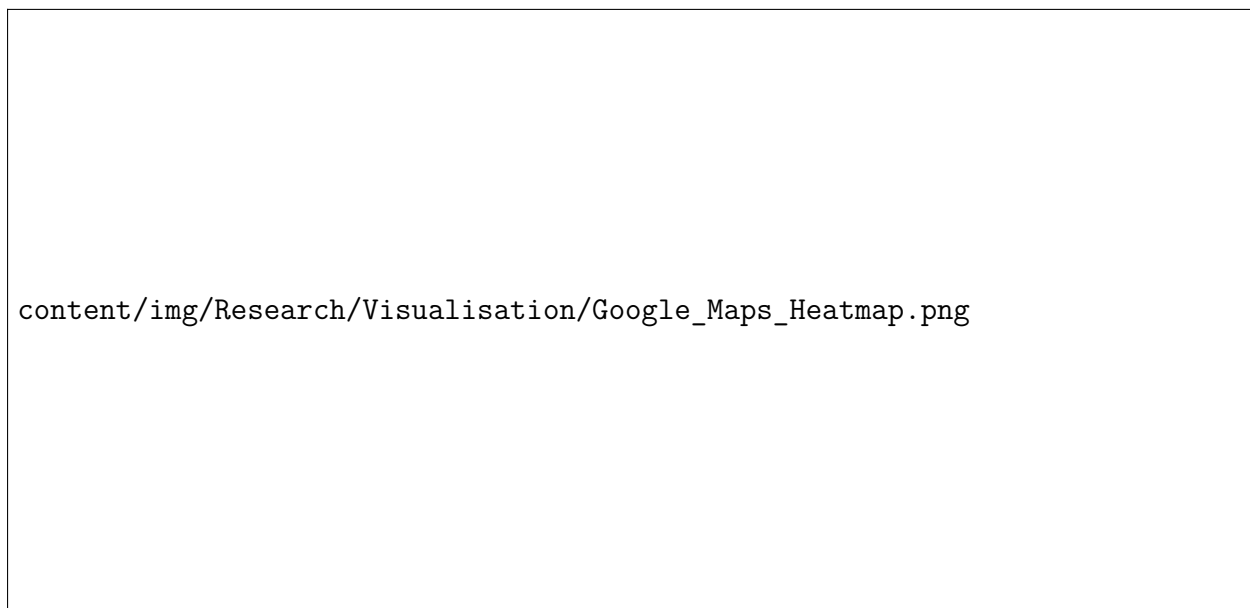


Abbildung 5.9.: Exemplarische geobasierte Heatmap aus der Google Maps Dokumentation

5.3. Bibliothek zur visuellen Darstellung insbesondere von Graphen

Nachdem die theoretischen Visualisierungsmethoden der bestehenden Literatur nun abschließend erläutert worden sind, wird nun eine Bibliothek für die Implementation der Darstellung von Graphen vorgestellt. *Cytoscape* ist eine Bibliothek für auf JavaScript-basierende Applikationen, welche die Erstellung eines Graphen im Frontend wesentlich erleichtert. Häufig wird Cytoscape für biologische Prozesse und Zusammenhänge verwendet, da besonders diese natürlichen wissenschaftlichen Bereiche ähnliche Strukturen wie Graphen aufweisen. Ursprünglich ist Cytoscape auch für genau diese Zwecke an der „University of Toronto“ entwickelt und anschließend in „Oxford Bioinformatics“ publiziert worden. Die Anwendungsbereiche von Graphenvisualisierungen gehen jedoch weit über den biomolekulare Bereich hinaus.

Cytoscape bringt einige Vorteile mit sich. Die Bibliothek ist einfach zu verwenden und verfügt über eine relativ gute Dokumentation, welche man unter dieser URL finden kann: <https://js.cytoscape.org/>. Darüberhinaus gibt es inspirierende Demonstrationen, wie man die Bibliothek verwenden kann. Diese Beispiel haben bei der Implementierung des Prototypen sehr geholfen.

¹Hierbei wird der Begriff „Heatmap“ allgemein verwendet, da diese nicht an ein Raster gebunden ist.

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

Bemerkenswert an Cytoscape ist auch das Eco-System und die Community, da es nicht nur viele Plugins, sprich Erweiterungen, gibt, sondern auch eine Variation an unterstützten Modulsystemen, darunter „ES modules“, „Node.js“ und „AMD/Require.js“. Des Weiteren wird Cytoscape in allen modernen Browsern unterstützt - eine extrem wichtige Eigenschaft für Frontend-Developer heutzutage. Und genau das sind auch einige der Gründe, warum ich mich für die Verwendung der Bibliothek Cytoscape entschieden habe.

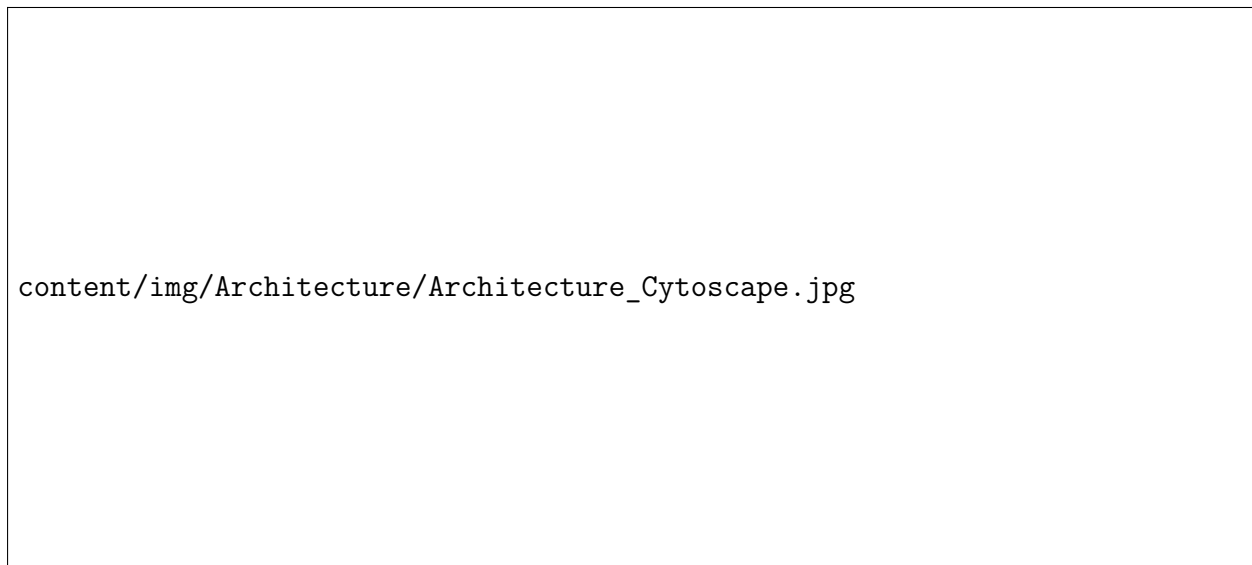


Abbildung 5.10.: Architektur unseres Prototypen mit Fokus auf die Bibliothek Cytoscape

Die Darstellungen anderer Grafiken, wie beispielsweise Diagramme, verwenden jeweils entsprechend passende Bibliotheken. *ng2-charts* eignet sich besonders gut für das Darstellen von Diagrammen in Angular Applikationen. Auch diese Bibliothek verfügt über eine hervorragende Dokumentation (<https://valor-software.com/ng2-charts/>).

6. Architektur des Prototypen

Unser Prototyp verarbeitet die Daten von Siemens auf eine performante Art und Weise. Zuerst werden die Fehlerdaten im ausfallsicheren *Messaging System* **Kafka** von Apache geschrieben. Diese Nachrichten werden anschließend vom *Backend*-System, welches aus einer relationalen Datenbank (**PostgreSQL**) und dem **Java Spring Framework** besteht, verarbeitet. Außerdem stellt das *Backend* diese Daten mittels **GraphQL-API** zur Verfügung. Somit kann das *Frontend*, welches mit **Angular** implementiert worden ist, die Daten via Abfragen visualisieren.

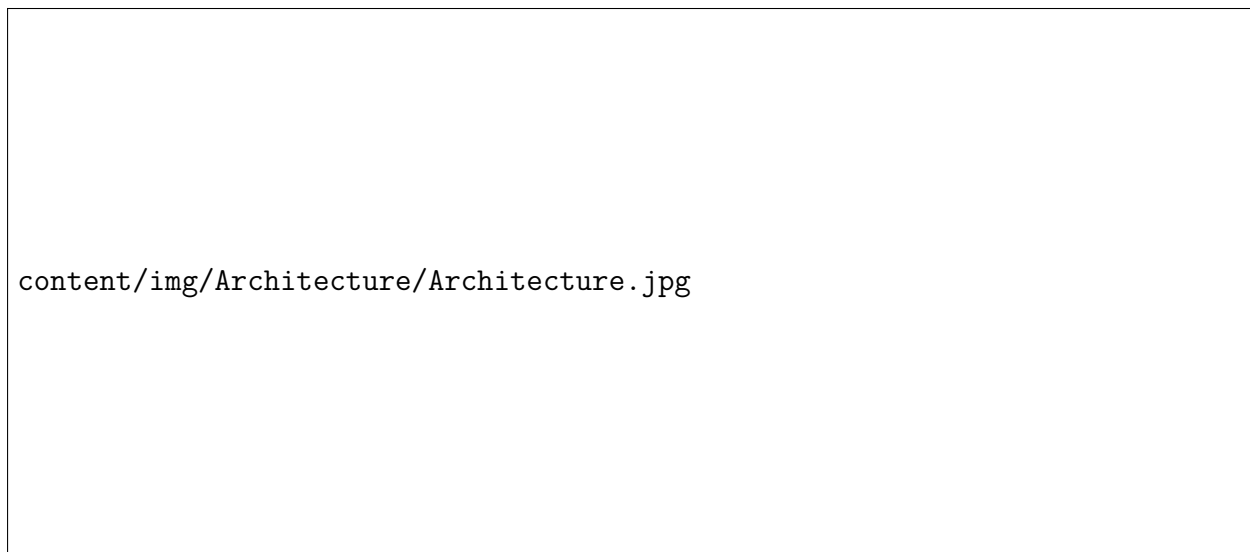


Abbildung 6.1.: Übersicht der Architektur unseres Prototypen

In diesem Kapitel werden die einzelnen Komponenten dieser *Service-Oriented-Architecture* (kurz SOA) im Detail beschrieben und erläutert.

6.1. Apache Kafka und das Backend

Die Implementierung dieses Systems basiert auf einer Architektur, die auf Apache Kafka, dem Java Spring Framework und PostgreSQL aufbaut. In diesem Kapitel wird der Ablauf der Datenverarbeitung von der Fehleranalyse bis zur Bereitstellung der Daten durch die GraphQL-API beschrieben. Ergänzend dazu werden die verwendeten Technologien im Detail erläutert und die Gründe für ihre Auswahl dargelegt.

6.1.1. Messaging Service: Apache Kafka

Die Daten werden zunächst durch die Siemens GNA Software, die für die Fehleranalyse zuständig ist, erfasst. Diese Daten werden kontinuierlich an ein Apache Kafka-Cluster gesendet, wie in der Abbildung 6.2 erkannt werden kann. Apache Kafka fungiert dabei als zentraler Datenstrom, der die Ereignisse oder Nachrichten empfängt und speichert. Hierbei gewährleistet Kafka eine zuverlässige und skalierbare Datenübertragung. Außerdem wird durch Apache Kafka eine starke Entkopplung zwischen der Siemens GNA Software und dem

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

Backend-System hergestellt. Das Backend profitiert vor allem von der hohen Datendurchsatzrate, welche Apache Kafka erreichen kann, denn nach jeder Auswertung der Fehler im Netzdatenmodell wird eine hohe Menge an Daten dem Backend übertragen. Die Technologie Apache Kafka ist schon genauer im Kapitel 3.3.1 beschrieben worden. Wie Apache Kafka aufgesetzt und in das *Backend* integriert worden ist, wird im Kapitel 7.1 genauer erklärt.



content/img/Architecture/Architecture_Kafka.jpg

Abbildung 6.2.: Architektur unseres Prototyps mit Fokus auf Kafka

6.1.2. Logik im Backend: Spring Framework

Das Backend-System, das mit dem *Java Spring Framework* entwickelt wurde, ist für die Verarbeitung der empfangenen Daten zuständig, dies ist auch in der Darstellung 6.3 zu sehen. Mithilfe von *Spring for Apache Kafka*, welches die Integration von *Kafka* in *Spring* ermöglicht, werden die Daten von Apache Kafka abgefragt und an das Backend weitergeleitet. In Spring werden diese Daten dann entsprechend verarbeitet und für die Speicherung in der Datenbank vorbereitet.

figures/krems	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT
	Krems
	Abteilung: Informationstechnologie



Abbildung 6.3.: Architektur unseres Prototyps mit Fokus auf das *Backend*

Die Entscheidung für *Java* als Programmiersprache und *Spring* als Application-Framework ist im Wesentlichen aus zwei Gründen gefällt worden. Einerseits ist die Verwendung von diesen Technologien eine fixe Anforderung seitens Siemens gewesen. Auf der anderen Seite bieten sie immense Vorteile für die Entwicklung der Anwendungen in Bezug auf unsere spezifischen Nutzungsszenarien. In den folgenden Abätzen werden diese Technologien vorgestellt und ihre Vorteile genauer beleuchtet.

Java ist eine objektorientierte, plattformunabhängige Programmiersprache, die ursprünglich von Sun Microsystems entwickelt wurde und jetzt von Oracle Corporation gepflegt wird. Sie wurde in den 1990er Jahren eingeführt und hat sich seitdem zu einer der beliebtesten Programmiersprachen entwickelt, insbesondere für die Entwicklung von Unternehmensanwendungen, Webanwendungen, mobilen Anwendungen und großen Systemen. Ein wichtiges Konzept von *Java* ist die objektorientierte Programmierung (OOP), das bedeutet, dass alles in *Java* als Objekt betrachtet wird. Objekte sind Instanzen von Klassen, die Daten und Methoden zur Manipulation dieser Daten enthalten. OOP-Konzepte wie Vererbung, Polymorphismus und Kapselung werden in *Java* stark unterstützt. Dadurch kann eine intuitive und effiziente Programmierung erreicht werden, was eine schnelle Entwicklung von Anwendungen ermöglicht. Ein großer Selling-Point dieser Programmiersprache ist die Plattformunabhängigkeit. *Java*-Programme werden in Bytecode kompiliert, der auf der *Java Virtual Machine (JVM)* ausgeführt wird. Dadurch sind *Java*-Anwendungen plattformunabhängig, was bedeutet, dass sie auf verschiedenen Betriebssystemen wie Windows, macOS und Linux ausgeführt werden können, solange eine *JVM* verfügbar ist. Ein weiterer Vorteil ist die Unterstützung von *Multi-Threading*, welches Anwendung erlaubt, parallel mehrere Aufgaben abzuarbeiten. Dies ist besonders in Anwendungsfälle, welche eine große Last auf Anwendungssysteme ausüben, zum Beispiel bei Server-Systemen von Vorteil. Außerdem hat *Java* ein reichhaltiges

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

Ökosystem an Erweiterungen, wie zum Beispiel *Frameworks* oder *Bibliotheken*. Dies liegt daran, dass *Java* eine sehr große Popularität im Enterprise-Sektor genießt. Denn die Unternehmen, welche *Java* verwenden, unterstützen die Entwicklung von diesen Erweiterungen, damit ihre Produkte verlässlich und effektiv funktionieren. Aus all diesen Gründen ist die Verwendung von *Java* in dem Fall dieser Arbeit optimal.

Java Spring Framework ist eine Erweiterung von *Java*, welche eine einfache Entwicklung von robuste, skalierbare und wartbare Anwendungen ermöglicht. Definitionsgemäß ist *Spring* ein *Application-Framework* und ein *Inversion of Control Container*. Ein *Application-Framework* ist ein strukturierter Satz von Bibliotheken, Komponenten und Tools, der die Entwicklung von Anwendungen erleichtert, indem er eine grundlegende Struktur und Funktionalität bereitstellt. Es werden komplexe Aspekte der Anwendungsentwicklung, welcher grundsätzlich bei jeder Applikation gleich ist, wie der Netzwerk- oder Datenbankzugriff, abstrahiert. Damit kann der Fokus der Entwickler auf die eigentliche Anwendungslogik gelenkt werden und Ressourcen für die Erstellung von redundanten Komponenten eingespart werden. Um Probleme bei der Wiederverwendung zu vermeiden, wird auf das Konzept der Standardisierung gesetzt, welche eine zuverlässige Nutzung durch klar definierte Regeln schafft. Mit *Inversion of Control* ist eine Entwicklungstechnik gemeint, welche ermöglicht, Objekte zu erstellen und ihre Abhängigkeiten dynamisch einzufügen. Dies fördert lose Kopplung zwischen den verschiedenen Komponenten einer Anwendung und erleichtert die Testung der Applikation. Das *Java Spring Framework* bietet in diesem Kontext, die Umgebung an, welche das dynamische Auflösen und Einfügen von Abhängigkeiten in den Anwendungskomponenten ermöglicht. Der bedeutendste Vorteil in der Verwendung von *Spring* in Bezug auf diese Arbeit ist, die leichte Integration von verschiedenen Technologien in die Anwendung durch die unzähligen Erweiterungen von diesem Framework. Beispielsweise ist die Einbindung der PostgreSQL Datenbank über die *Spring Data JDBC* Bibliothek mit Leichtigkeit geschaffen worden, eine genauere Erklärung darüber ist im Kapitel 7.2. Des Weiteren wird *Kafka for Spring Framework* für das Interagieren des *Backend-Systems* mit Apache Kafka verwendet, welche auch eine Erweiterungsbibliothek ist, eine detaillierte Erläuterung dazu ist im Kapitel 7.1.

6.1.3. Persistierung der Daten: PostgreSQL Datenbank

Die verarbeiteten Daten werden in einer *PostgreSQL*-Datenbank persistiert. *PostgreSQL* bietet eine robuste und leistungsstarke relationale Datenbanklösung, die die Anforderungen an die Datenspeicherung erfüllt. Das *Java Spring* Backend kommuniziert über *Spring Data JDBC* mit der Datenbank, um die Daten effizient zu speichern und abzurufen. Der ausschlaggebendste Grund zur Wahl von *PostgreSQL* ist die Anforderung von Siemens, dass eine relationale Datenbank verwendet werden soll, welche schon dem kooperierenden Entwicklungsteam von Siemens bekannt ist. Ein weiterer Grund ist die starke Popularität unter den Entwicklern und die starke Etablierung am Markt seitens *PostgreSQL*. In den nächsten Absätzen wird *PostgreSQL* und seine Vorteile genauer erklärt.

PostgreSQL ist ein leistungsstarkes Open-Source-Datenbankmanagementsystem (DBMS), das auf dem objektrelationalen Datenbankmodell basiert. Außerdem unterstützt *PostgreSQL* Transaktionen, die es ermöglichen, eine Gruppe von Operationen als eine einzige atomare Einheit auszuführen. Das *ACID-Prinzip* (*Atomicity, Consistency, Isolation, Durability*) wird unterstützt, was die Datenintegrität sicherstellt. Ein weiterer Vorteil von *PostgreSQL* ist,

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

dass es in der Lage ist, als Hochverfügbarkeitsdatenbank zu operieren. Denn es können Daten zwischen mehreren Datenbankservern repliziert werden und somit kann bei einem Ausfall von einem DB-Server ein anderer die Arbeit übernehmen. Relationale Datenbanken haben ein sehr einfaches Konzept. Es ist aufgebaut auf Tabellen, welche Relationen zueinander haben, das heißt es können Zeilen aus verschiedenen Tabellen in Verbindung stehen, wobei jede Zeile eine Sache beschreibt. Beispielsweise könnte einer Person eine Adresse zugeordnet werden durch eine Relation zwischen einer Personentabelle und Adresstabelle. Jedoch haben relationale Datenbanken einen Haken, wenn die Daten zu stark vernetzt sind, dann kommt diese Technologie an seine Grenzen. Im Falle dieser Arbeit sind die zu verwaltende Daten Graphdaten, was stark vernetzte Daten sind, wie trotzdem eine effiziente Speicherung und performantes Abfragen gelungen ist, wird im Kapitel 7.2 beschrieben.

6.2. GraphQL und das Frontend

Diese Sektion erklärt kurz die technische Implementierung des Frontends mit zugehörigen *Services* in der Architektur. Wenn man sich Abbildung 6.4 noch einmal ansieht, kann man erkennen, dass die Daten über die *GraphQL API* vom Frontend abgerufen werden können. Dieses ist mit *Angular* implementiert. Für die Visualisierung des Graphen wird die Bibliothek *Cytoscape* verwendet.



Abbildung 6.4.: Architektur unseres Prototypen mit Fokus auf das *Frontend*

6.2.1. Die Schnittstelle zwischen Backend und Frontend: GraphQL

Eine populäre Schnittstelle für Daten, welche eine Form von Graphstruktur aufweisen, ist *GraphQL*. Die Besonderheit dieser *API* ist die deklarative Natur der Schnittstelle. Im Gegensatz zu REST, gRPC und anderen *APIs* bestimmt bei *GraphQL* der *Client*, welche Eigenschaften und Felder er erhalten möchte. Dadurch wird die Transportmenge an Daten über das Internet reduziert.

Um im Folgenden die besonderen Eigenschaften von *GraphQL* näher zu erläutern, wird nun ein Beispiel eingeführt. Dabei hat ein Benutzer einen Benutzernamen und eine normalisierte Adresse. Da diese Daten aus der realen Welt am besten in einem Graphen anstatt einer Tabelle abgebildet werden können, sieht man in Abbildung 6.5 diese Abstraktion der Daten.

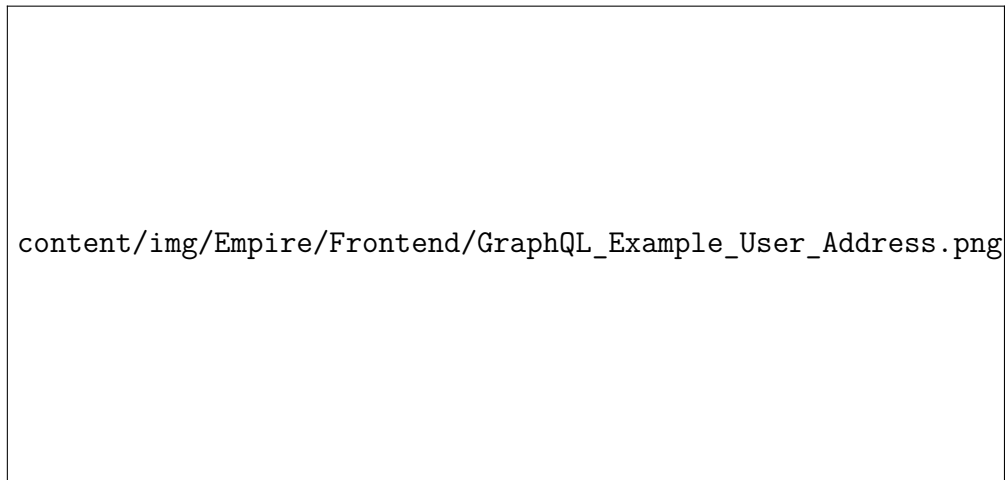


Abbildung 6.5.: Speicherung von Daten unseres Beispiels in einem Graphen

GraphQL hat den weiteren Vorteil, dass die Anzahl an Abfragen, welche durchgeführt werden müssen, um alle notwendigen Informationen zu erhalten, erheblich reduziert wird. Viele der herkömmlichen *APIs* benötigen eine Abfrage für die Basisinformationen (Benutzernamen) und anschließend eine weitere Abfrage für die Adresse. Dies rührt daher, weil Daten aus der realen Welt meistens eine hohe Komplexität aufweisen, und in relationalen Datenbanken in verschiedenen Tabellen gespeichert werden. *GraphQL* bietet Möglichkeiten, um alle Informationen, welche den Benutzer betreffen in einer Abfrage abzuhandeln. Der *Client* nimmt sich dann nur die Daten, die für den speziellen *Use-Case* gebraucht werden.

Außerdem definiert man bei *GraphQL* zuerst ein Schema, welches laut *GraphQL*-Dokumentation wie eine „*Shared Language*“, also geteilte Sprache, sein soll. Das gesamte Business Modell soll mittels natürlicher Sprache beschrieben werden können und genau so im Schema verwirklicht werden. Beim „*schema-first-approach*“ wird dieses Schema als aller erstes definiert, sodass *Backend* und *Frontend* darauf aufbauend gleichzeitig programmiert werden können.

Und genau das sind die Gründe, warum wir uns für *GraphQL* entschieden haben. Im Gegensatz zum *Backend*, wo das Java Spring Framework vorgeschrieben war, weil viele Systeme und Applikation von Siemens und auch allgemein auf der gesamten Welt mittels Java laufen und demnach die Wartbarkeit höher ist, wenn die Ingenieure keine neue Sprache erlernen müssen, hatten wir bei der *API* die freie Auswahl.

GraphQL basiert auf „*Queries*“ und „*Mutation*“, wobei eine *Query* immer nur zum Abfragen von Daten dient und eine *Mutation* Daten am Server beziehungsweise in der Datenbank verändert. Die Schönheit von *GraphQL* liegt in der Einfachheit dieser Abfragen:

```

1 query {
2   user (input: {id: 69}) {
3     __typename
4     ... on UserResult {
5       username
6       address {
7         street
8         streetNumber
9         postalCode

```

```

10     city
11   }
12 }
13 ... on Error {
14   cause
15 }
16 }
17 }

```

Quellcode 6.1: eine einfache *GraphQL*-Abfrage

Für die verschiedenen Typen dieser Abfrage sind noch viele weitere Felder, wie zum Beispiel die Kategorie, Schweregrad und auch der Type des Findings verfügbar. Jedoch benötigt der Client in diesem Fall dieser Werte nicht, weswegen die Daten nicht übertragen werden müssen.

```

1 {
2   "data": {
3     "user": {
4       "__typename": "UserResult",
5       "result": {
6         "username": "trueberryless",
7         "address": {
8           "street": "An Der Bundesstrasse",
9           "streetNumber": 4,
10          "postalCode": "6642",
11          "city": "Hinterhornbach",
12        }
13      }
14    }
15  }
16 }

```

Quellcode 6.2: das Ergebnis einer einfachen *GraphQL*-Abfrage

Mithilfe dieser *API* werden die Daten aus dem *Backend* nun zur Verfügung gestellt. Dabei unterstützt die *API* eine Menge an Schnittstellen, welche in der Abbildung 6.6 eingesehen werden können. Alle diese Abfragen sind für die Implementierung des *Frontends* notwendig und äußerst nützlich. Dabei muss wieder bedacht werden, dass nicht unbedingt Weise alle Felder aller Abfragen in Verwendung treten, da bei *GraphQL* der *Client* entscheidet, welche Daten er erhalten möchte.

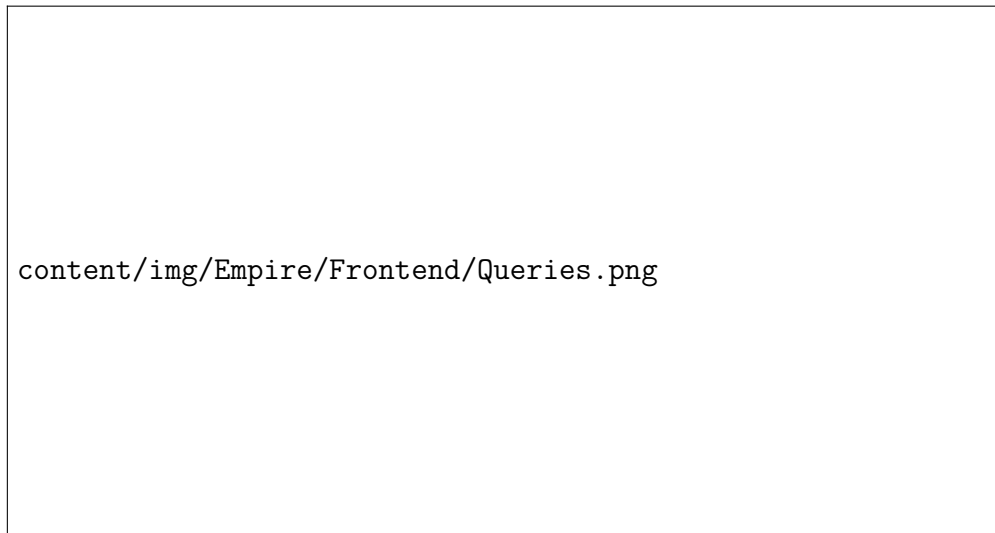


Abbildung 6.6.: Auflistung der Abfragen, welche mittels *GraphQL* implementiert sind [Schema Definition]

6.2.2. Das Framework aus der JavaScript Welt: Angular

Das von Google entwickelte, auf der Programmiersprache *TypeScript* aufbauende Framework „*Angular*“ ist erstmals 2009 als Nebenprojekt von Misko Hevery und Adam Abrons am Markt erschienen (damals noch *AngularJS*) und wird aktiv seit 2016 von Google weiterentwickelt (*Angular*¹). Erst im November 2023 ist Angular rebranded worden und hat ab Version 17 nun auch ein cooles Logo. In Abbildung 6.7 kann man dieses wunderbare Logo auch bewundern:

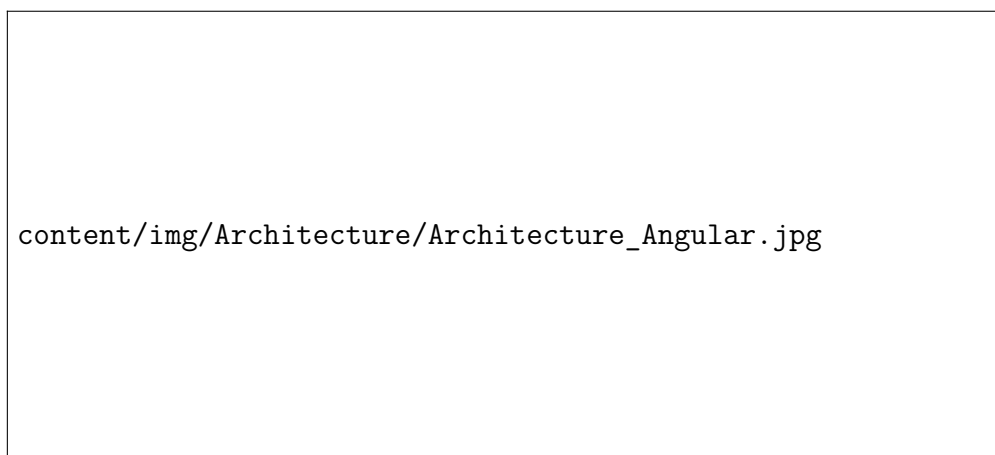


Abbildung 6.7.: Architektur unseres Prototypen mit Fokus auf das Frontend, welches mit Angular implementiert ist

Google wirbt für das *open-source Framework* mit Schnelligkeit, Ausfallsicherheit aufgrund leichter Skalierbarkeit und einer unfassbar starken *Community*. Jedoch haben wir uns aus keinem dieser bemerkenswerten Gründe für dieses Framework entschieden. Siemens hat uns wärmstens empfohlen, dass wir Angular verwenden sollen, wie man auch im Plichtenheft im Anhang sehen kann. Das liegt daran, dass Siemens in Zukunft die Implementierung der

¹*Angular* ist nicht gleich *AngularJS*.

figures/krems	<div>HÖHERE TECHNISCHE BUNDES - LEHRANSTALT</div> <div>Krems</div> <div>Abteilung: Informationstechnologie</div>
---------------	--

Webapplikation wahrscheinlich, jedoch noch nicht zu 100%, mit Angular durchführen will. Unsere Applikation wäre somit eine der ersten, welche das *Corporate Design* von Siemens in einer echten Angular-Anwendung umsetzt.

Eines der bemerkenswertesten Features bei Angular ist die direkte Unterstützung der *CLI* (*Command Line Interface*), sodass alle Aktionen mittels Terminal durchgeführt werden können. Diese Aktionen umfassen das Erstellen eines Projekts, das Hinzufügen von Bibliotheken, das Erstellen eines Komponenten bis hin zum Starten der Applikation. Summa summarum ist die *CLI* von Angular ein sehr umfangreiches, hilfreiches, mächtiges und von der *Community* geschätztes Tool, um das Angular *Framework* zu steuern und es hilft auch vor allem Programmierern, welche Angular neu erlernen, da die Einfachheit der Befehle nicht nur gut dokumentiert, sondern auch intuitiv sind.

Außerdem hat Angular bereits einige moderne Features *out of the box*. Es unterstützt *Server-Side Rendering*, ein Prozess, bei welchem bereits auf dem Server die gesamte HTML-Dateien gerendert werden und der Benutzer nicht auf extra *JavaScript* und CSS-Dateien warten muss, was die Ladezeiten deutlich verbessert. Angular ist zusätzlich kompatibel mit den beliebtesten CSS-Präprozessoren, wie zum Beispiel SCSS, SASS, LESS und einige mehr. Präprozessoren sind in diesem Sinn verbesserte und erweiterte Versionen des Grundgerüsts von CSS, welche das Schreiben dieser Beschreibungssprache mit unterschiedlichen Regeln erheblich erleichtern. Bevor der Browser diese Präprozessor-Daten anschließend jedoch interpretieren kann, müssen diese in reines CSS umgewandelt werden. Aus diesem Grund auch der Name „Präprozessor“.

Darüber hinaus hat Angular bereits ein eigenes Testsystem und Routing-System. Es kann mittels einfacher Installation des `@angular/pwa` Pakets zu einer *Progressive Web App* gemacht werden und ist kompatibel mit unzähligen *JavaScript*-Bibliotheken auf dem Markt, wie zum Beispiel die ebenfalls von Google entwickelte *UI*-Bibliothek „Angular Material“, welche Standardkomponenten, wie zum Beispiel *Slider*, *Toggle*, *Button*, *Datepicker* usw. mit sich bringt. Die Möglichkeiten für Webapplikationen mit Angular sind heutzutage unbegrenzt.

Angular basiert auf Komponenten. Dies sind kleinere, spezifische Teile des *Frontends*, welche innerhalb von anderen Komponenten oder unter gewissen Routen angezeigt werden können. Der Vorteil von Komponenten ist die Wiederverwendung des Codes, da jeder Komponent beliebig oft angezeigt werden kann. Jeder Komponent hat eine eigene Verwaltung der für diesen Komponenten relevanten Logik, ein eigenes CSS-Styling und HTML natürlich. Dadurch wird automatisch eine Art Entkoppelung der Logik im *Frontend* gewonnen. In einem „Tic-Tac-Toe“-Spiel könnte man zum Beispiel einen Komponenten für ein einzelnes Kästchen programmieren, welches die Logik für das Klicken dieses Feldes implementiert.

In unserem Fall wird Angular nun verwendet, um die Daten des Stromnetzwerkes veranschaulicht darzustellen. Dazu nutzt das *Frontend* die *GraphQL*-Schnittstelle, um geschickt an die Daten zu kommen. Beispielsweise wird nur eine einzige *Query* durchgeführt, um alle Daten in der Tabelle in Abbildung 6.8 zu bekommen. Wundern Sie sich nicht: Die Daten sind anonymisiert.

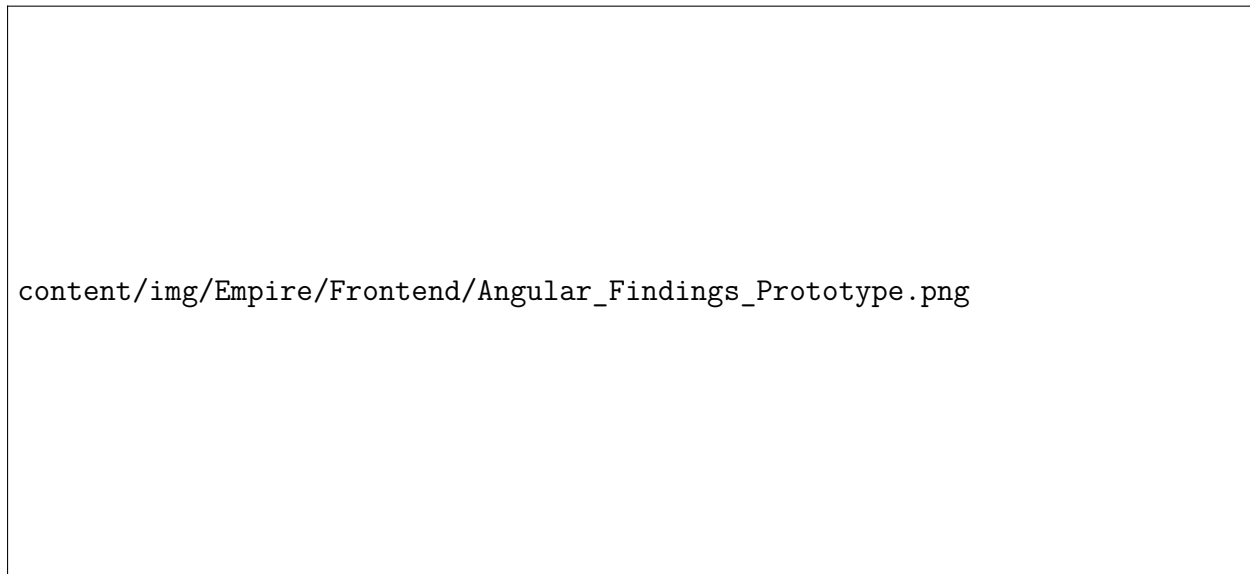


Abbildung 6.8.: Abbildung der Tabelle der Fehlerdaten im Stromnetzwerk mit Sortier- und Filteroptionen unseres Prototypen

Wie man sehen kann erhält man zu jedem aktuell auftretenden Fehler im Netzmodell von Siemens Informationen bezüglich deren Typ, Klasse, Schweregrad und Kategorie. Außerdem hat jeder Fehler eine Nachricht, einen Namen und mehrere Identifikatoren, welche in der Abbildung 6.8 unter dem ausgewählten Objekt (erkennbar an der größeren Höhe der Zeile) gefunden werden können. All diese Informationen können dank *GraphQL* in einer Abfrage gesammelt und angezeigt werden, was den Prozess des Erstellens dieses Komponenten deutlich erleichtert.

6.2.3. Die Bibliothek zum Erstellen des Graphes: Cytoscape

Aufgrund der unendlichen Möglichkeiten und vielfältigen *npm*-Unterstützung, zählt auch die *Cytoscape*-Bibliothek zu einem unglaublich leicht zu integrierenden Tool zur Visualisierung von Graphen. Die Bibliothek bietet eine umfassende Dokumentation aller Funktionen. Das Grundkonzept der Bibliothek basiert auf einer Liste von Elementen, welche die Knoten und Kanten gesammelt umfasst. Diese Knoten und Kanten werden über die Bibliothek in einem konfigurierbaren Layout in einen *Canvas* im HTML angezeigt. *Cytoscape* unterstützt dabei bereits alle notwendigen Animations- und Interaktionsmöglichkeiten für Maus und Finger (bei Touchdisplays).

Unser Prototyp implementiert einen Graphen, welche die Objekte des Stromnetzwerkes (Umspannwerk, Leiter, Schalter, Widerstände, Generatoren, Verbraucher, ...) als Knoten in Verbindung setzt. Um diesen Graphen initial zu erstellen, werden mittels *GraphQL* die gesamten Objekte mit der Information über dessen Verbindungen zu anderen Objekten geladen und anschließend zu Knoten und Kanten aufbereitet (Quellcode 6.3,) sodass *Cytoscape* diese im Graphen rendern kann.

```

1 getNodeEdges(input: ModelObjectInput): Observable<GraphElement[]> {
2   return this.modelObjectService.getModelObjects(input).pipe(
3     switchMap((data) => {

```

```

4      var modelObjectResult: ModelObjectResult = data;
5
6      var nodes: GraphElement[] = [];
7      var edges: GraphElement[] = [];
8
9      /* calculate nodes and edges logic */
10
11     return of([...nodes, ...edges] as GraphElement[]);
12 }
13 );
14 }

```

Quellcode 6.3: Erstellen der Knoten und Kanten mittels Liste von Objekten von *GraphQL*

Diese Liste von Graph-Elementen wird anschließend an die *Cytoscape*-Bibliothek übergeben. Somit verwaltet nun *Cytoscape* die Objekte und wie diese im HTML *Canvas* gerendert werden.

```

1 this.graphService.getNodeEdges(/* input */).subscribe((data) => {
2   this.nodesEdges = data as GraphElement[];
3
4   this.cy = cytoscape({
5     container: document.getElementById('cy'),
6     elements: this.nodesEdges as GraphElement[],
7     style: this.stylesheet as Stylesheet[],
8     layout: colaLayout,
9   });
10 });

```

Quellcode 6.4: Laden der Knoten und Kanten in den Graphen mittels *Cytoscape*

Wie man im Quellcode 6.4 außerdem sehen kann, wird das „cola“-Layout verwendet, um die Knoten effizient im *Viewport* der Webseite zu positionieren. „Cola“ ist hierbei eines der Layouts, welches physikalische Gesetze aus der Natur nutzt, um die Kräfte zwischen den einzelnen Knoten zu berechnen und um diese gleichmäßig im *Canvas* zu verteilen. Viele weitere in *Cytoscape* integrierten Layouts, wie zum Beispiel „cise“, „cose-bilkent“, „fcose“, „d3-force“ und „euler“, basieren ebenfalls auf dieser Eigenschaft, sogenannte „*force-directed*“. Der Grund für die Auswahl des „cola“-Layouts ist die Funktionalität der unendlichen Ausführung (die Berechnungen der wirkenden Kräfte untereinander werden niemals gestoppt), welche den Graphen immer cool aussehen lässt.

7. Backend-System

Das Backend, entwickelt im Java Spring Framework, übernimmt die Verarbeitung der empfangenen Daten. Mithilfe von *Kafka for Spring Framework* werden die Daten von Apache Kafka abgerufen und an das Backend übergeben. Danach werden die Daten in eine *PostgreSQL* Datenbank persistiert, mittels dem *Spring Data JDBC* Module von dem *Spring Framework*. Damit die Daten auch verwendet werden können, werden sie über eine *GraphQL* Schnittstelle zur Verfügung gestellt. In den kommenden Kapiteln wird über die Probleme, welche während der Implementierung aufgetreten sind, geschrieben. Zu jeder Problematik wird auch der gewählte Lösungsansatz vorgestellt.

7.1. Apache Kafka und Java Spring Framework

Apache Kafka ist ein open-source *Messaging System*, welches in dieser Arbeit für die entkoppelte Datenübertragung von der Siemens GNA Software zu dem Backend, eingesetzt wird. *Kafka* ist für die Anwendung in großen verteilten System konzeptioniert, welche sehr hohe Anforderungen an einer Reihe von Aspekten, wie zum Beispiel Ausfallsicherheit oder Skalierbarkeit haben. Dadurch wird *Apache Kafka* ein sehr mächtiges, aber auch komplexes *Messaging System*, welches in der Einrichtung und Verwendung sehr aufwendig sein kann. In diesem Kapitel wird beschrieben, wie durch die Nutzung der richtigen Techniken und Tools dieser Aufwand fast völlig eliminiert worden ist.

7.1.1. Einrichtung von Apache Kafka

Die Architektur von jeder *Apache Kafka* Installation besteht aus mindestens einer *Zookeeper* Instanz und einer *Kafka* Instanz. Damit die Einrichtung stark vereinfacht wird, nutzt der Prototyp die *Apache Kafka* Distribution für *Docker*, entwickelt von dem Unternehmen Confluent.

Docker ist eine Open-Source-Plattform, die es Entwicklern ermöglicht, Anwendungen in sogenannten *Containern* zu erstellen, zu verwalten und auszuführen. Container sind leichte, tragbare und isolierte Umgebungen, die alle notwendigen Abhängigkeiten enthalten, um eine Anwendung auszuführen, einschließlich des Betriebssystems, der Laufzeitumgebung, Bibliotheken und anderer Konfigurationen. Docker vereinfacht die Bereitstellung von Anwendungen, indem es eine konsistente Umgebung über verschiedene Systeme hinweg bereitstellt und eine schnellere Entwicklung und Bereitstellung ermöglicht.

Docker Compose ist ein Tool, das es ermöglicht, mehrere Docker-Container als eine einzige Anwendung zu definieren und zu verwalten. Mit *Docker Compose* können Entwickler eine YAML-Datei verwenden, um die Konfiguration mehrerer Container zu definieren, einschließlich ihrer Abhängigkeiten, Netzwerkeinstellungen und anderer Konfigurationsoptionen. *Docker Compose* erleichtert die Lokalisierung und Bereitstellung von komplexen Anwendungen, die aus mehreren miteinander verbundenen Containern bestehen, und ermöglicht eine einfache Wartung dieser Anwendungen.

Im Kontext des Prototyps ermöglicht *Docker* eine verlässliche Installation von *Apache Kafka* auf jeder Art von Umgebung, welche *Docker* selbst installiert haben. Dadurch kann man diese Distribution von *Kafka* auf Linux, Windows oder auch macOS installieren, ohne befürchten

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

zu müssen, dass Fehler durch diese unterschiedlichen Umgebungen entstehen. *Docker Compose* schafft die Komplexität, welche *Apache Kafka* durch seine verteilte und ausfallsichere Natur hat, in einer deklarativen Art zu minimieren. Diese Einfachheit kann in dem Quellcode 7.1 erkannt werden, denn *Zookeeper* und *Apache Kafka* können in einer übersichtlichen Weise konfiguriert werden. Ein weiterer Vorteil ist, dass die ganze Konfiguration in einem einzigen File definiert ist, welches eine einfache Installation auf mehreren Systemen ermöglicht.

```

1 version: '3'
2 name: 'ndfa-backend'
3 services:
4   zoo:
5     image: confluentinc/cp-zookeeper:latest
6     hostname: zoo
7     ports:
8       - "127.0.0.1:2181:2181"
9     environment:
10      ZOOKEEPER_CLIENT_PORT: 2181
11      ZOOKEEPER_SERVER_ID: 1
12      ZOOKEEPER_SERVERS: zoo:2888:3888
13
14   kafka:
15     image: confluentinc/cp-kafka:latest
16     hostname: kafka
17     ports:
18       - "127.0.0.1:9092:9092"
19       - "127.0.0.1:9999:9999"
20     environment:
21      KAFKA_ADVERTISED_LISTENERS: INTERNAL://kafka:19092,EXTERNAL://${
22 DOCKER_HOST_IP:-127.0.0.1}:9092
23      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INTERNAL:PLAINTEXT,EXTERNAL:
24 PLAINTEXT
25      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
26      KAFKA_ZOOKEEPER_CONNECT: "zoo:2181"
27      KAFKA_BROKER_ID: 1
28      KAFKA_LOG4J_LOGGERS: "kafka.controller=INFO,kafka.producer.async.
29 DefaultEventHandler=INFO,state.change.logger=INFO"
30      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
31      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
32      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
33      KAFKA_JMX_PORT: 9999
34      KAFKA_JMX_HOSTNAME: ${DOCKER_HOST_IP:-127.0.0.1}
35      KAFKA_AUTHORIZER_CLASS_NAME: kafka.security.authorizer.AclAuthorizer
36      KAFKA_ALLOW_EVERYONE_IF_NO_ACL_FOUND: "true"
37     depends_on:
38       - zoo

```

Quellcode 7.1: Docker Compose File zur einfachen Einrichtung von *Apache Kafka*

7.1.2. Verwendung von Apache Kafka im Java Spring Framework

Um eine einfache und abstrakte Interaktion mit *Apache Kafka* zu ermöglichen, nutzt der Prototyp das *Java Spring Framework* Modul *Kafka for Spring Framework*. Damit kann eine nahtlose Integration von *Kafka* in das Spring Framework erfolgen und eine einfache

Entwicklung von Systemen, welche mit dem *Apache Kafka Messaging System* funktionieren, möglich gemacht werden.

Die vollständige Konfiguration von der Backend-Applikation bezüglich *Kafka* kann in der globalen Konfigurationsdatei von *Spring* erfolgen, das ist sehr vorteilhaft, weil so leicht die Übersicht behalten werden kann. In dem Quellcode 7.2 ist ein Auszug dieser Datei, welcher sich auf *Kafka* fokussiert.

```

1  kafka:
2      bootstrap-servers: "${KAFKA_BOOTSTRAP_SERVERS:localhost:9092}"
3      consumer:
4          auto-offset-reset: earliest
5          group-id: "ndfa"
6          key-deserializer: org.apache.kafka.common.serialization.
StringDeserializer
7          value-deserializer: org.springframework.kafka.support.serializer.
JsonDeserializer
8          max-poll-records: 10000

```

Quellcode 7.2: Auszug aus der Konfigurationsdatei für *Spring*

Ein weiterer Aspekt der Konfiguration, welcher nicht in der globalen Konfigurationsdatei passiert, ist die Definition von *Topics*, das sind Kanäle, auf denen Nachrichten geschickt und von *Consumer* konsumiert werden können. In dem Code 7.3 kann leicht erkannt werden, wie einfach die Erstellung von *Topics* mit der Hilfe von *Kafka for Spring Framework* ist. Im Falle des Prototyps sind zwei Kafka-Themen in Verwendung. Das "Findings"-Topic ist zuständig für die Übertragung der gefundenen Fehler und das zweite wird für die Übermittlung des Stromnetzmodells benötigt.

```

1  @Bean
2  public KafkaAdmin.NewTopics createTopics() {
3      return new KafkaAdmin.NewTopics(
4
5          TopicBuilder.name("model_objects")
6              .build(),
7          TopicBuilder.name("findings")
8              .build()
9      );
10 }

```

Quellcode 7.3: Definieren eines Topics mithilfe von *Kafka for Spring Framework*

Ein sehr wichtiger Teil bezüglich der Interaktion mit *Apache Kafka*, ist die Erstellung von *Consumer*, denn sie können die Nachrichten von *Topics* auslesen. Durch die Unterstützung des *Spring* Modul kann mit einer Leichtigkeit, *Consumer* erstellt werden. Außerdem können spezielle Konfigurationen auch umgesetzt werden, beispielsweise, wie es im Code 7.4 zu erkennen ist, kann leicht *Batch-Processing* beim Auslesen umgesetzt werden. Des Weiteren kann man auch eine gleichzeitige Auslesung von *Topics* oder die Konvertierung von dem *Message Body* in Java Klassen eingestellt werden. Im Prototyp hat das "Findings"-Topic und das "ModelObject"-Topic beide Batch-Processing aktiviert, damit eine schnelle Verarbeitung der Daten ermöglicht wird. Das Kafka-Thema, welches für die Modellobjekte des Stromnetzes zuständig ist, kann sogar mit mehreren Threads das Topic parallel auslesen, da die Reihenfolge der Nachrichten keine Rolle spielt.

```

1      @KafkaListener(topics = "model_objects", batch = "true", concurrency = "3"
2      ,
3      properties = { "spring.json.value.default.type=com.siemens.backend
4      .model.gna.GnaModelObject" }
5      )
6      public void processMessage(List<GnaModelObject> content) throws
7      SQLException {
8          logger.debug("received %s GnaModelObjects on topic model_objects".
9      formatted(content.size()));
10         modelObjectsService.insertAndDeleteGnaModelObjects(content);
11     }

```

Quellcode 7.4: Definition eines Consumers mithilfe von *Kafka for Spring Framework*

Es wurde in den vorherigen Absätzen bestätigt, dass die Verwendung von *Kafka for Spring Framework*, eine einfache Integration von *Apache Kafka* in den Java Code schaffen kann. Durch die starke Abstrahierung können sogar Entwickler, welche mit diesen *Messaging System* nicht so viel am Hut haben, auch programmieren.

7.2. PostgreSQL

PostgreSQL ist eine objektrelationale Datenbank, welche für die Persistierung der Daten, die über *Apache Kafka* erhalten worden sind, zuständig ist. Jedoch beschreiben diese Daten ein Stromnetzwerk, welches von Natur aus einer Graphenstruktur ähnelt. Graph-Daten sind immer stark vernetzt und somit problematisch für die Speicherung in relationale Datenbanken. In diesem Kapitel wird genauer diese Problematik beschrieben und der gewählten Lösungsweg vorgestellt.

7.2.1. Problematik der Speicherung von Graph-Daten in einer relationalen Datenbank

Eine relationale Datenbank organisiert Daten in Tabellen, wobei jede Tabelle eine Sammlung von Zeilen und Spalten darstellt. Jede Zeile in einer Tabelle entspricht einem Datensatz, während jede Spalte ein Attribut oder eine Eigenschaft dieses Datensatzes darstellt. Die Struktur des relationalen Datenbankmodells wird durch eine Reihe von Regeln definiert, die sicherstellen, dass die Daten konsistent sind. Ein einfaches Beispiel wäre die Zuordnung von mehreren Schülern zu einem Lehrer. Dafür muss eine Lehrertabelle und Schülertabelle erstellt werden und zusätzlich in der Schülertabelle eine weitere Spalte hinzugefügt werden, welche die IDs der Lehrer beinhaltet, um eine Verbindung aufzubauen. Diese Spalte wird auch die Fremdschlüsselspalte genannt, denn es wird ein fremder Schlüssel (ID von Lehrer), also eine ID, welche nicht von demselben Datensatz herkommt, in die Zeile eingefügt.

Ein offensichtliches Problem bei dieser Art Verbindungen aufzubauen ist, wenn man eine N:M Relation umsetzen möchte. Solch eine Relation in den Kontext des Beispiels würde bedeuten, dass Schüler mit mehreren Lehrern und Lehrer mit mehreren Schülern in Verbindung stehen. Jedoch ist es nicht möglich, diesen Fall nur mit einer Fremdschlüsselspalte umzusetzen, denn dafür müsste man mehrere IDs in dieser Spalte einfügen, was aber das grundsätzliche Konzept der Atomarität der Spalten verletzt. Um diese Relation dennoch umzusetzen, muss eine

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

weitere Tabelle hinzugefügt werden, welche in jeder Zeile eine Verbindung zwischen einem Lehrer und einem Schüler definiert. Diese weitere Tabelle fügt einen weiteren Schritt für der Auflösung der Beziehungen hinzu, was eine schlechtere Abfrageperformance zur Folge hat.

Ein weiteres Problem zeigt sich bei der Einfügung der Daten in die Tabellenstruktur. Denn normalerweise wird eine Fremdschlüsselspalte durch eine Regel geschützt, welche nur Schlüssel von einer bestimmten Tabelle erlaubt. Will jedoch eine Zeile hinzugefügt werden, welche einen noch nicht eingefügten Datensatz referenziert, ist das nicht möglich und der Datensatz muss zwischengespeichert werden. In dem Fall dieser Arbeit ist dieses Problem bedeutend, da die Daten in kleine Stücke über *Apache Kafka* erhalten werden und deren Reihenfolge nicht definiert wurde. Außerdem ist das Berechnen der richtigen Reihenfolge von großem Aufwand, insbesondere wenn man die Größe der Daten, welche das Stromnetzwerk beschreiben, bedenkt. Das Problem prägt sich aus durch eine überaus lange Zeit, welche benötigt wird, um die Daten vollständig einzufügen.

Damit diese zwei Probleme gelöst werden konnten, ist ein Lösungsansatz herausgearbeitet worden, welcher in dem folgenden Kapitel erläutert wird.

7.2.2. Lösungsansatz für effiziente Speicherung von Graph-Daten

Dieser Lösungsansatz verfolgt zwei verschiedene Konzepte, um die Problematik zu lösen. Einerseits, die Vereinfachung der Tabellenstruktur und andererseits die Auflösung aller Regeln bezüglich der Fremdschlüssel in Tabellen.

Anstatt eine komplexe Struktur von Tabellen aufzubauen, wird im Prototypen eine einfache Struktur angestrebt. Jedoch sollte auch angemerkt werden, dass Daten durch einen komplexen Aufbau der Tabellen leichter konsistent gehalten werden können. Aber im Kontext dieser Arbeit wird dieser Vorteil zum Nachteil, weil das Datenmodell muss in der Lage sein, jegliche Fehler des Stromnetzmodelles darzustellen. Die finale Struktur der Tabellen kann in der Abbildung 7.1 betrachtet werden. Es wird auf jede überflüssige Tabelle verzichtet und somit können performante Abfragen erreicht werden.

In der Darstellung kann auch erkannt werden, dass es keine Verbindungen zwischen den Tabellen gibt, welche eine Relation andeuten. Aber tatsächlich gibt es für jedes *Finding* (Fehler in Netzmodell) ein oder mehrere *Modelobjekte*, welche die eigentlichen Objekte des Stromnetzes beschreiben, beispielsweise könnte ein *Modelobjekt* ein Transformator oder eine Hochspannungsleitung darstellen. Zusätzlich sind natürlich die Objekte des Stromnetzes miteinander verbunden, dafür wird die *Connections* Tabelle benötigt, denn sie stellt jede Verbindung mit einer Zeile dar. Diese Verbindungen werden nicht angezeigt, weil sie durch kein Regelwerk definiert worden sind. Durch den Verzicht auf diese Regeln können die Datensätze direkt in die Datenbank eingefügt werden und die Zeit für die Befüllung der Datenbank nach Änderung des Stromnetzmodelles kann minimiert werden.

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie



Abbildung 7.1.: Tabellenstruktur des Prototyps

8. Webapplikation Angular

Der Prototyp verfügt über eine verlässliche, schnelle Webapplikation, welche mit *Angular* entwickelt worden ist. *Angular* ist ein von Google entwickeltes *Framework* zur einfachen Erstellung von skalierbaren und performanten Webapplikationen. Die Erlernung der Grundlagen des *Frameworks* ist im Vergleich zu anderen *JavaScript-Frameworks* deutlich einfacher. Die Grundprinzipien umfassen das Verständnis eines *Components*, das *Routing* von *Angular* bei einer *Single Page Application* und *Dependency Injection* im Allgemeinen. Da die detaillierten Funktionalität von *Angular* bereits im Architekturkapitel besprochen worden sind, gehen wir hier nicht mehr näher darauf ein. Stattdessen wird dieses Kapitel die Funktionalität unserer Webapplikation im Näheren behandeln.

Das Auffälligste bei diesem Teil des Prototypen ist die konsistente Umsetzung des *Corporate Designs* von Siemens. Die ikonische türkise – manche Menschen würden auch einfach blau oder grün dazu sagen – Unternehmensfarbe zieht sich dank *Angular Theme* durch die gesamte Webapplikation. Angefangen beim Logo, über die Anwendungsleiste am oberen Bildschirmrand bis hin zu den Ladebalken und ähnlichem ist alles in dem typischen Siemens-Türkis gehalten. Diese Entscheidung ist bewusst getroffen worden, da die Farbe sofort auf das Unternehmen selbst referenziert. Einen Teil dieses Designs dürfen Sie in Abbildung 8.1 genießen.

content/img/Empire/Frontend/Angular_Sneak_Peak.png

Abbildung 8.1.: Sneak Peak unserer Angular Applikation

i In der Entwicklungsphase des Prototypen sind anonymisierte Testdaten verwendet worden, da Clemens und Felix rechtlich gesehen keine Kundendaten sehen dürfen.

8.1. Dashboard mit Kennzahlen des Netzmodells

Kennzahlen in einem *Dashboard* attraktiv, benutzerfreundlich und intuitiv darzustellen, ist selbst für fortgeschrittene Designer oft ein langwieriger Prozess. Eine einfache Art, die Dauer für die Schaffung eines übersichtlichen *Dashboards* zu minimieren, ist die Verwendung von vorgefertigten *UI-Elementen*. Diese Elemente können aus beliebigen Bibliotheken stammen. Für *Angular* eignet sich am besten die Bibliothek **ng2-charts**.

Die Bibliothek *ng2-charts* ist im Prinzip eine Schnittstelle zwischen *Angular* und einer universellen *Javascript-Diagramm-Bibliothek* namens *Chart.js*. Diese unterstützt wiederum eine Vielzahl an Diagrammen, von Linien- über Flächen bis hin zu Donatdiagrammen. Für das *Dashboard* unseres Netzwerkanalyseprogrammes habe ich ausschließlich Donatdiagramme verwendet, um die prozentuellen Verteilungen der verschiedenen Eigenschaften,

content/img/Empire/Frontend

Abbildung 8.2.:
Donatdiagramm

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

darunter der Schweregrad, die Kategorie, das Spannungslevel und die Klasse beziehungsweise der Typ des Fehlers, einheitlich darzustellen.

In Abbildung 8.2 ist ein Beispiel eines Donatdiagrammes gegeben. Dieses wird im *Dashboard* verwendet, um die Verteilung der beiden Kategorien darzustellen.

Diese Bibliothek wird nun verwendet, um vier Donatdiagramme auf dem Dashboard darzustellen. Diese Diagramme geben den Ingenieuren von Siemens einen schnellen Überblick über grundlegende Eigenschaften der aktuellen Fehler im Netzmodell. Jedes der vier Diagramme stellt die prozentuelle Verteilung bezüglich einer Eigenschaft dar. Diese vier Eigenschaften sind: Klass, Kategorie, Schweregrad und Spannungslevel. Wie man in Abbildung 8.3 sehen kann, sind alle vier Eigenschaften sehr „einseitig“. Mit dieser Einseitigkeit ist die klare Dominanz beziehungsweise die Vorherrschaft des Hauptsegmentes der jeweiligen Eigenschaft gemeint. Wie man an den Legenden sehen kann, sind beispielsweise fast alle Fehler im Netzmodell mit Schweregrad „CRITICAL“ kategorisiert worden (der große gelbe Anteil im dritten Diagramm in Abbildung 8.3).



Abbildung 8.3.: Dashboard des Prototypen

Bei diesen Diagrammen wird auch ein hoher Wert auf die Interaktivität gelegt. Wie man beim ersten Diagramm in Abbildung 8.3 auch sehen kann, gibt es eine dem Mauszeiger folgende Informationen bezüglich des Anteils dieses ausgewählten Segments. Dieser sogenannte „*Tooltip*“ ist äußerst intuitiv, da er heutzutage schon an vielen Stellen in der Webentwicklung auftritt.

Um die Interaktivität noch mehr zu steigern, ist bei *ng2-charts* auch die Funktion implementiert, bestimmte Anteile im Diagramm dynamisch auszublenden. Bei schlauem Anwenden dieser Funktion, können bereits sehr nützliche Informationen aus einigen Diagrammen abgelesen werden, wie man in Abbildung 8.4 sieht. Man sieht zum Beispiel, dass ein Großteil der

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

gefilterten Fehler bei Stromkreisunterbrechern und Knoten entsteht (Diagramm 1) und nur sehr wenige Fehler eine Spannung von 400V aufweisen.

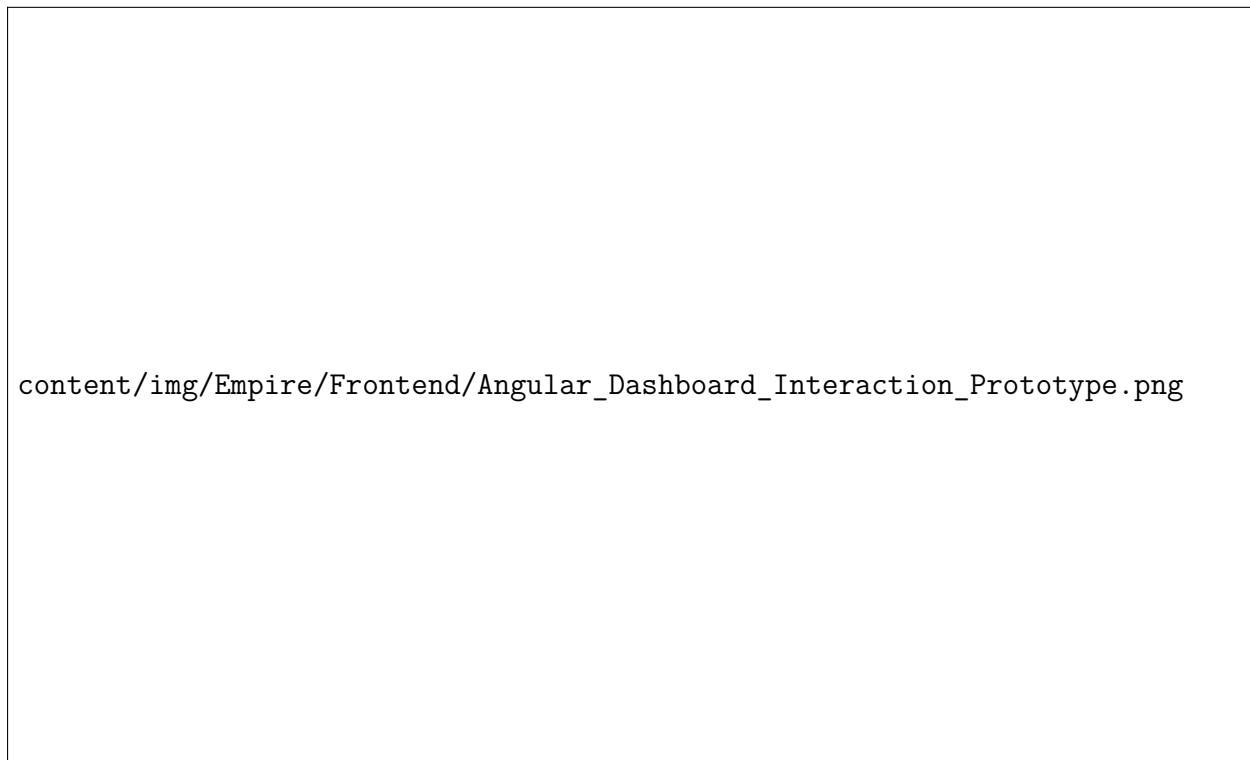


Abbildung 8.4.: Interaktion mit den Diagrammen des Dashboards

Summa summarum ist das Dashboard ein sehr effizienter Weg, um einen schnellen Überblick über das aktuelle Geschehen im Netzmodell zu erhalten und bestimmte Eigenschaften im allgemeinen Kontext zu interpretieren. Die Diagramme sind intuitiv zu lesen und mit einigen Interaktionen ausgestattet. Außerdem ist das Dashboard eine wichtige Komponente ein moderner Applikationen, da diese immer eine Art „*Landingpage*“ beziehungsweise Einstiegsseite für die Benutzer ist und diesen herzlich willkommen heißt.

8.2. Tabellen mit Filter- und Suchfunktionen

Die Applikation verfügt neben dem Dashboard auch über eine Tabelle mit Filter-, Such- und Sortierungsmöglichkeiten. Im Prinzip ist diese Tabelle eine verbesserte Darstellung der von Siemens verwendeten Log-Dateien, da interaktiv mit den Ergebnissen gearbeitet werden kann. Sie zeigt standardmäßig alle gefundenen Fehler im Netzmodell an. Dieser Fehler beziehen sich beispielsweise auf kaputte Schalter, Generatoren oder Leitungen. Allerdings können Fehler auch eine größere Bedeutung im Stromnetzwerk, wie zum Beispiel ein Umspannwerk oder eine Gruppe von Elementen, beinhalten. Mithilfe der Filtermöglichkeiten können somit schwerwiegende Fehler schneller gefunden werden. Dazu kommen wir jedoch später noch. Zuerst sehen wir uns in Abbildung 8.5 die Standardansicht der Tabelle an.

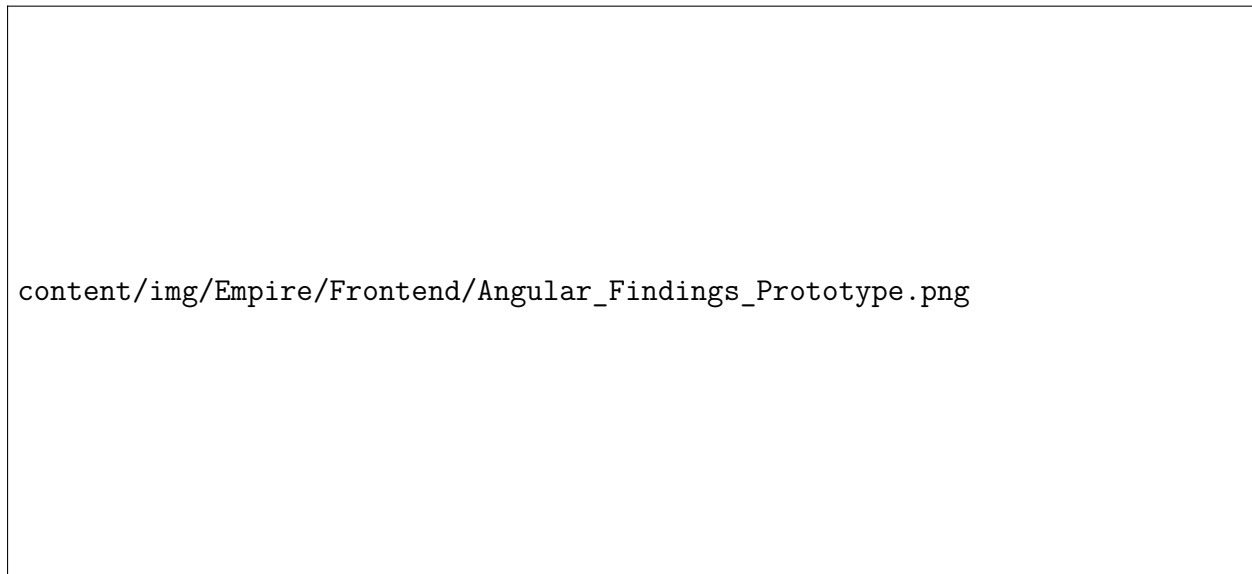


Abbildung 8.5.: Tabelle mit den aktuellen Fehlern im Stromnetzwerk

8.2.1. Sortierung

Ein wichtiger Bestandteil moderner Tabellen in der digitalen Welt ist die Möglichkeit einer Sortierung. Diese Implementation ist dank der Bibliothek, welche in unserem Prototypen für alle *UI*-Elemente verwendet wird, „Angular Material“ wirklich einfach. Nach Einstellung der Art der Sortierung – Diese kann variieren. Beispielsweise werden bei numerischer Sortierung Zahlen logisch der Größe nach geordnet, während alphabetisches Sortieren bei Zahlen keinen Sinn ergibt, da somit „111“ vor „22“ kommen würde. – und richtiger Handhabung der Aktualisierung der Daten, sobald eine neue Spalte ausgewählt wird, funktioniert diese wirklich verlässlich. Diese Art der Implementierung ist auch als „deklarative Programmierung“ bekannt, da man dem *Framework* nur sagt, *was* zu tun ist. Im Gegensatz zur „imperativen Programmierung“ müssen nicht die einzelnen Schritte selbst implementiert werden – *wie* etwas zu tun ist.

In Abbildung 8.6 kann man sehen, dass die Datensätze alphabetisch aufsteigend nach dem Typen geordnet sind, wie auch an dem kleinen Pfeil neben dem Spaltennamen erkennbar ist.

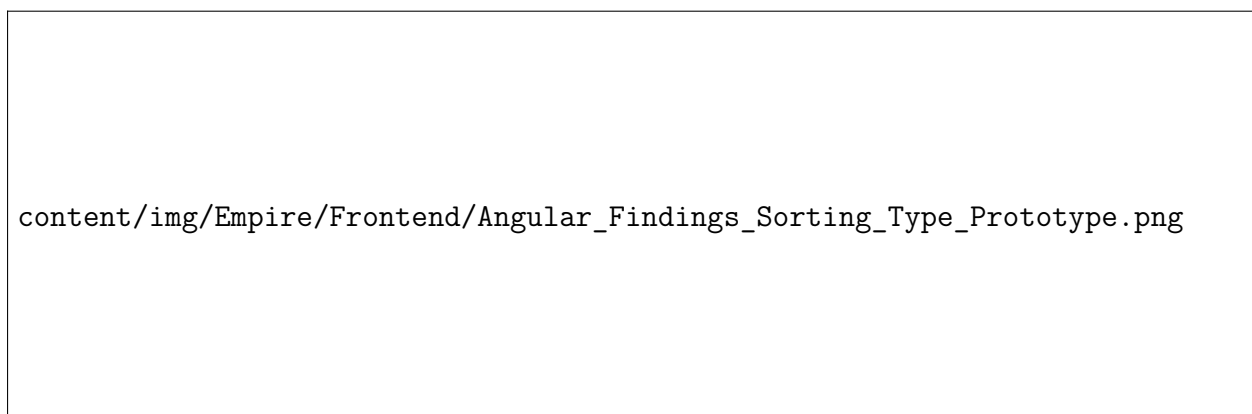


Abbildung 8.6.: Sortierungsoptionen der Tabelle des Prototypen

8.2.2. Filterung

Unsere Webapplikation verfügt außerdem über einige Filteroptionen. Dazu gehört natürlich ein Eingabefeld, welches über alle Spalten der Tabelle sucht und anschließend nur relevante Ergebnisse anzeigt. In Abbildung 8.7 sieht man beispielsweise die Filterung nach dem Wort „Node“, welches bei den ersten drei Ergebnissen in der Klasse und ansonsten auch immer in der Nachricht des Eintrages vorkommt. Dieses Eingabefeld ist laut Siemens auch besonders wichtig, um schnell nach bestimmten Fehlern mit einer konkreten *ID* suchen zu können. Hierbei kann man diese Identifikatoren zwar nicht als eigene Spalte sehen, jedoch sind sowohl die *GUID* des Fehlers als auch die *GUID* des Objektes, wo dieser Fehler im Stromnetzwerk auftritt, unter den zusätzlichen Informationen sichtbar. In Abbildung 8.5 hat man bereits diese Zusatzinformationen einsehen können.

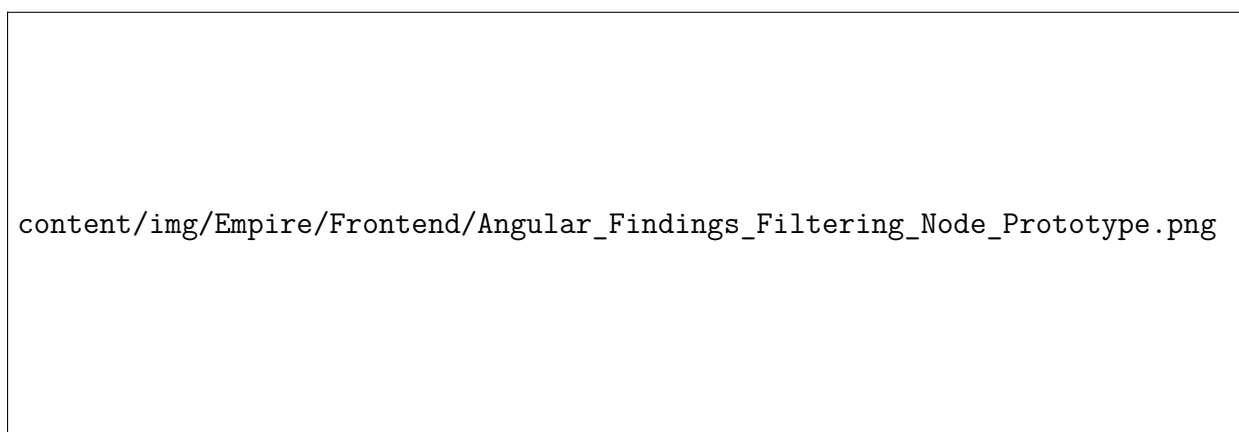


Abbildung 8.7.: Filterungsoption mit Eingabefeld über alle Spalten

Im Laufe des Projektes ist noch die Anforderung hinzugekommen, die Einschränkung der Daten nach bestimmten Kriterien filter zu können. Hierbei handelt es sich genau wie beim *Dashboard* 8.1 um die vier Eigenschaften: Klasse, Kategorie, Schweregrad und Spannungslevel. Im untigen Beispiel 8.8 ist die Einschränkung der Netzwerkfehler des Schweregrads abgebildet.

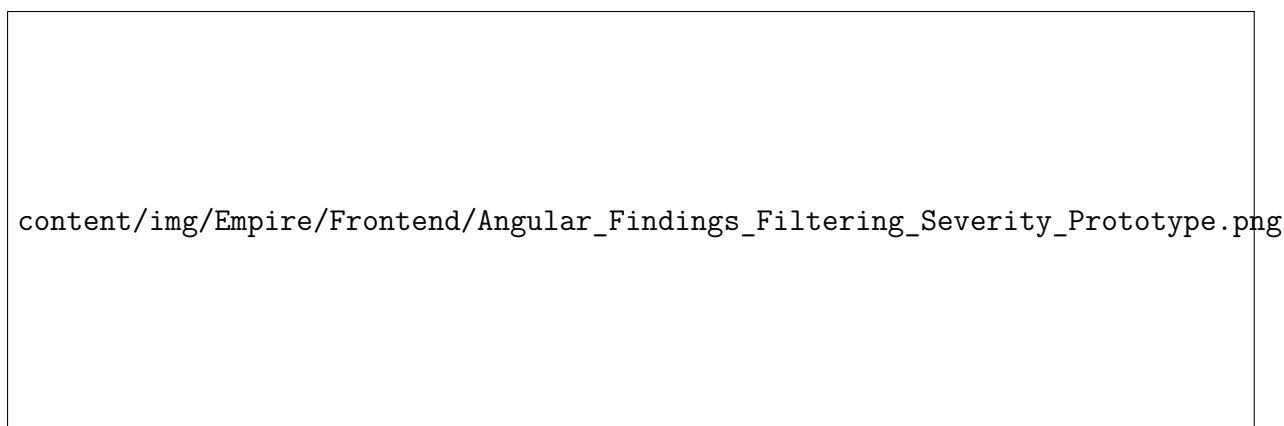


Abbildung 8.8.: Filterungsmöglichkeit mit Dropdown-Menü für Schweregrad

Eine wichtige Komponente der Filterungsmöglichkeiten ist der „Reset“-Button. Dieser erlaubt das schnelle Zurücksetzen aller Filter, falls der Benutzer das Gefühl hat, sich in den

Tiefen der Filterungsoptionen verirrt zu haben. Trotz seiner Einfachheit hat der *Reset-Button* in vielen Fällen eine hohe Daseinsberechtigung, da er intuitiv und äußerst nützlich ist.

8.2.3. Paginierung

Im Stromnetzwerk treten stetig einige Fehler auf, da das Netzmodell konstant umgebaut und verbessert wird. Eine *GraphQL*-Abfrage, welche all diese Daten auf einmal anzeigen wollen würde, führt zwangsmäßig zu langen Ladezeiten. Aus diesem Grund hat heutzutage jeder Anwendungsfall, welcher mit vielen Daten arbeitet, irgendeine Art Pagination eingebaut. Dabei wird einfach nur ein bestimmter Teil der Daten geladen. Andere Datensätze können nach Bedarf nachgeladen werden. Während viele *Social Media*-Plattformen auf *Lazy Loading* setzen – hierbei werden neue Daten beim Scrollen nachgeladen – verwendet unsere Applikation die von *Angular Material* ausimplementierte Pagination. Der Nutzer kann hierbei einfach auf die beiden Pfeile rechts oben klicken, um weitere Einträge zu laden. Um die Orientierung über die Daten zu behalten, werden außerdem ständig Informationen bezüglich den momentan angezeigten Elementen dargestellt. Beispielsweise lässt sich der Nutzer in Abbildung 8.9 gerade die Elemente 351 bis 375 anzeigen. Insgesamt verfügen unsere Testdaten¹ über 45.249 Datensätze.

content/img/Empire/Frontend/Angular_Findings_Pagination_Prototype.png

Abbildung 8.9.: Paginierung der Tabelle mit den Fehlern

8.2.4. Besonderheit: Spannung

Einige Datensätze verfügen über die zusätzliche Information des Spannungslevels. Dieses wird dynamisch unter den zusätzlichen Informationen angezeigt und außerdem kann man nach diesen Werten filtern, da sie eine große Bedeutung im Stromnetzwerk haben. Aufgrund von Zeitmangel ist sich die Implementation der rekursiven Abfragen nach dem Spannungslevel nicht mehr ausgegangen. Das Netzmodell ist nämlich vereinfacht geschrieben in einer Baumstruktur aufgebaut. Teilweise haben Container höher in dieser Struktur Informationen bezüglich der Spannung des gesamten Bereiches. Jedoch hätte bei der Implementierung dieser Funktionalität die Rekursion zur Auflösung des Baummodells eine komplexere Datenabfrage zur Folge gehabt, weshalb auf dieses Feature verzichtet worden ist. Jedoch könnte dieses Leistungsmerkmal zukünftig noch erweitert werden.

Der anonymisierte Eintrag „Giant Cyclops X_24“ in Abbildung 8.10 verfügt zum Beispiel über ein Spannungslevel von 220kV.

¹Wir verwenden für unsere gesamte Applikation anonymisierte Testdaten, welche über das Apache Kafka System eingespielt werden, weil wir während der Entwicklungsphase aufgrund von Datenschutzgründen keine Kundendaten verwenden dürfen.

figures/krems	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

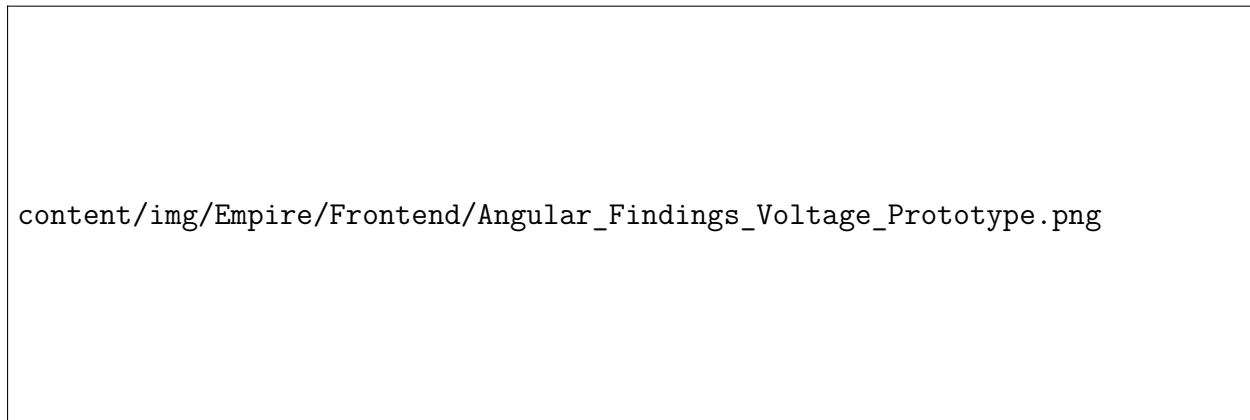


Abbildung 8.10.: Einträge, welche über einen Spannungswert von 220.0kV verfügen

Wie man in dieser Abbildung (8.10) auch schön sehen kann, verfügt jeder Eintrag über einen „Go to Graph →“-Button. Wohin dieser den Benutzer führt, wollen wir uns als nächstes ansehen.

8.3. Graph des Stromnetzmodells

Die Visualisierung eines gesamten Stromnetzwerkes auf einem kleinen Computerbildschirm ist nicht ganz so trivial, wie die Auflistung der aktuellen Fehler im Netzmodell. Dank der Bibliothek *Cytoscape* ist die Implementierung eines Graphen in einer Webanwendung erheblich leichter. Dennoch haben wir uns viele Gedanken machen müssen, wie wir diesen Teil des Prototypen gestalten wollen.

Die grundlegende Frage bei der Visualisierung von diesen spezifischen Daten ist: Sollen die relevanten Informationen im Graph als Knoten oder Kanten abgebildet werden? Standardmäßig geht ein Benutzer davon aus, dass die Knoten mehr Relevanz haben, als die Kanten, da letztere nur das Verbindungsstück zwischen den wesentlichen Elementen darstellen. Doch in manchen Szenarien, wie bei einem Stromnetzwerk zum Beispiel, können die Informationen auch in den Kanten enthalten sein. Immerhin ist fast jedes Element in unseren Daten nur mit exakt zwei Endpunkten verknüpft. Das ruft förmlich nach den Eigenschaften der Kanten.

Nichtdestotrotz haben Clemens und Felix sich dazu entschieden, die Elemente des Stromnetzwerkes einfach zu Knoten „heraufzustufen“, da dieser Ansatz viel intuitiver für die Ingenieure von Siemens ist. Aus diesem Grund haben die meisten Knoten in Abbildung 8.11 auch nur zwei anliegende Kanten zu anderen Knoten.

Ein weiterer Vorteil der Darstellung der Elemente in den Knoten des Graphen ist die erleichterte Visualisierung der Namen dieser Modellobjekte sowie intuitivere Interaktionsmöglichkeiten mit dem Benutzer. Denn viele Benutzer wissen, dass Knoten in einem Graphen auswählbar sind, während die Selektion einer Kante eher befremdlich wirkt.

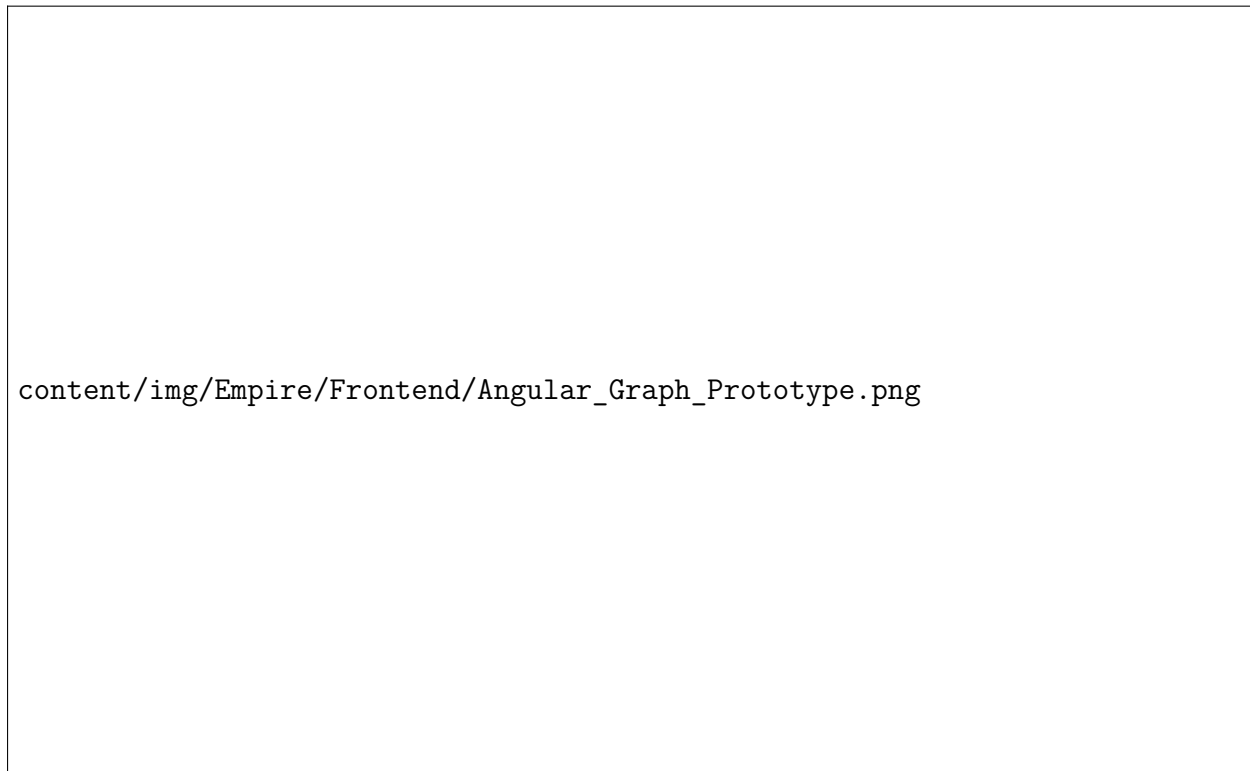


Abbildung 8.11.: Visualisierung eines Teils des Stromnetzwerkes in einem Graphen

Eine weitere Überlegung dieser Visualisierungsmethode ist der Einstiegspunkt beziehungsweise der Knoten, welcher als erstes geladen wird, gewesen. Denn aufgrund der schieren Menge an Elementen im Stromnetzwerk ist es einfach unmöglich alle Daten auf einmal zu laden. Aus diesem Grund wird immer nur ein Stück beziehungsweise ein kleiner Teil des Stromnetzwerkes angezeigt. Dabei ergibt sich jedoch die Frage: Welcher Teil des Netzwerks soll angezeigt werden, wenn nicht das gesamte Netzwerk angezeigt werden kann? Anders formuliert benötigen wir ein Startelement, von welchem aus die weiteren angeschlossenen Elemente geladen werden können. Somit haben wir die Möglichkeit, das Laden der Elemente auf eine gewisse Tiefe zu reduzieren. Folgendermaßen wird automatisch nur ein Teil des Graphen geladen, was wiederum die Schnelligkeit der Visualisierung deutlich erhöht.

Aus diesen genannten Gründen ist es bei unserer Applikation nicht möglich, den Graph ohne Startelement zu visualisieren. Und genau deswegen kann man den Graphen auch nur ansehen, wenn man auf den „Go to Graph →“-Button bei der Fehlertabelle klickt. Denn somit haben wir sichergestellt, dass der Graph mit einem fehlerhaften Objekt im Netzmodell startet, dessen nähere Umgebung man angenehm analysieren kann. In Abbildung 8.12 kann man sehr gut erkennen, dass das Startelement im Graphen auch blau markiert ist, damit die Siemens Ingenieure effizient und effektiv den Ursprung des Fehlers im Netzmodell ausfindig machen können.

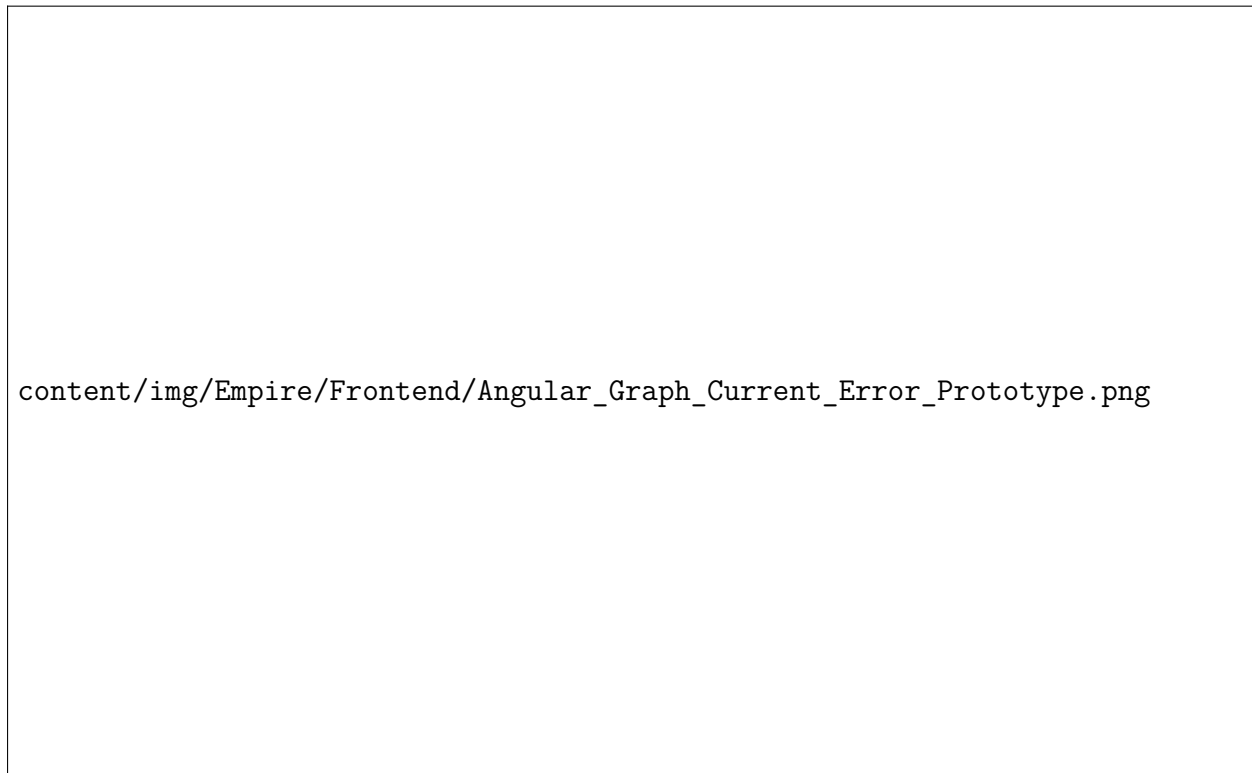


Abbildung 8.12.: Aktuelles fehlerhaftes Netzmodellobjekt im Graphen blau markiert

In dieser Abbildung (8.12) sieht man auch trotz Anonymisierung der Daten, dass zwischen zwei „aktiven“ Elementen (Schalter, Transformatoren, Widerstände, Generatoren, Sicherungen, usw.) meistens ein „passives“ Element (hauptsächlich Leiter) liegt. Aktive Elemente beschreiben in dem Kontext eher Elemente, welche Eigenschaften des Stroms verändern, während passive diesen nur transportieren. Wie man sehen kann, kommen diese beiden Gruppen – aktive und passive Elemente – abwechselnd hintereinander im Netzmodell vor: „Hobbit-hole“ – „Yak-52“ – „Hobbit-hole“. Noch besser ist diese besondere Eigenschaft in Abbildung 8.13 zu erkennen, wo sich „Yak“ und „Dori“ alternierend entlang der gelb markierten Verbindung ergänzen.

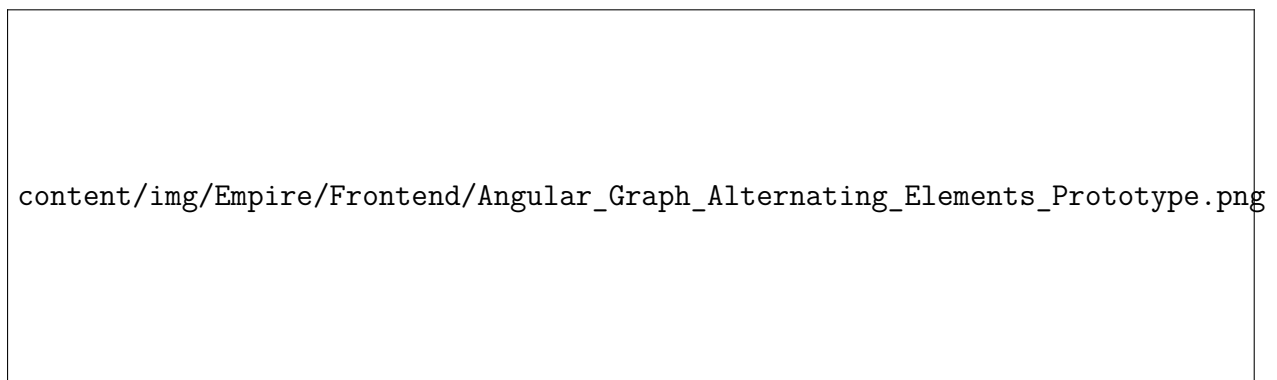


Abbildung 8.13.: Abwechselnde Vorkommnis der verschiedenen Elementgruppen im Stromnetzwerk

Im Sinne des Graphen ist auch anzumerken, dass die Positionierung der Knoten keine geo-

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT
	Abteilung: Krems Informationstechnologie

grafische Bedeutung haben, da uns diese Art von topografischen Daten nicht zur Verfügung steht. Stattdessen werden die Knoten mithilfe des *cola*-Layouts möglichst „effizient“ angeordnet. Effizienz in diesem Sinne bedeutet, dass sich die Knoten, wären sie Atome, in einer möglichst ruhigen, entspannten Position und angenehmen Entfernung von anderen Knoten befinden. Dieses Layout unterstützt dementsprechend die Darstellung eines natürlichen biologischen Systems, da sich beispielsweise auch Zellen im menschlichen Körper gleichmäßig in den Blutbahnen verteilen.

8.4. Applikation im Dark Mode

Zu guter Letzt sehen wir uns noch die Applikation im *Dark Mode* an. Diese Funktionalität ist meiner Meinung nach bei fast allen Webanwendungen ein Muss und genau aus diesem Grund wird sie logischerweise auch bei unserer Webapplikation unterstützt. Dank *Angular* in Kombination mit *Angular Material* ist die Umsetzung eines dunklen Modus wirklich nicht schwierig und zieht sich konsistent durch das gesamte Design durch. Dabei ist natürlich wichtig, dass die *Corporate Identity* von Siemens nicht verloren geht, weshalb die Hauptfarbe wieder das ikonische Türkis ist. In den folgenden Abbildungen 8.14, 8.15 und 8.16 können Sie Ihre Blicke über den angenehm zu erblickenden *Dark Mode* ruhen lassen, um sich von den vorherigen Abbildung zu erholen...

content/img/Empire/Frontend/Angular_Dashboard_Prototype_Dark.png

Abbildung 8.14.: *Dashboard im Dark Mode*

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

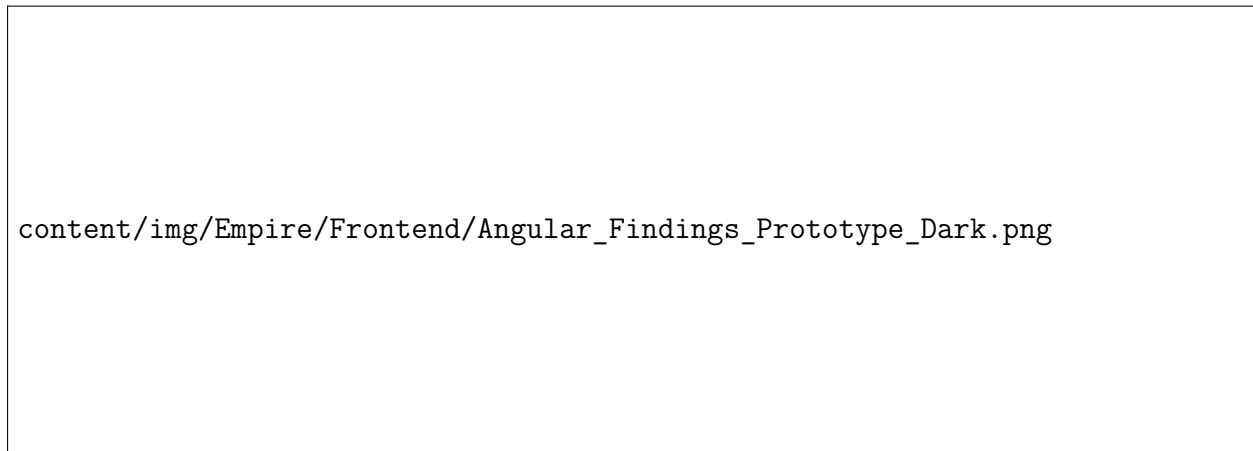


Abbildung 8.15.: Ergebnisse des Netzmodells im *Dark Mode*



Abbildung 8.16.: Visualisierung des Stromnetzmodells mittels Graph im *Dark Mode*

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

9. Bewertung

9.1. Enterprise Service Bus

9.2. Visualisierung des Netzmodells

10. Zusammenfassung und Ausblick

10.1. Zusammenfassung

Zusammenfassend war diese Diplomarbeit ein sehr lehrreiches Projekt, bei dem wir viele neue Erfahrungen gemacht haben. Neben der Tatsache, dass sowohl Clemens Schlipfinger und Felix Schneider neue Technologien, wie zum Beispiel Kafka, Spring und Angular, kennengelernt haben, haben die beiden auch ihre Teamfähigkeit, Flexibilität und den Umgang mit Problemen verbessern können.

Wir hoffen, dass Siemens in Zukunft das fertige Produkt in die interne Nutzung integrieren kann und somit das Finden von Fehlern im Stromnetzwerk effizienter gestalten kann. Falls nicht, sind wir trotzdem stolz auf unser Projekt allgemein und die ausgezeichnete Zusammenarbeit mit Siemens AG Österreich. Danke, dass ihr diese Diplomarbeit ermöglicht habt!

Es ist uns außerdem ein besonders wichtiges Anliegen, nochmals ein großes Dankeschön an unseren Betreuer und gleichzeitig auch Lehrer in dem Fach „Einführung in das wissenschaftliche Arbeiten“ Ing. Jürgen Katzenschlager MSc, BEd auszusprechen. Durch seine Unterstützung, Ratschläge und ständig kritisches Feedback hat sich die Diplomarbeit allgemein stetig verbessert und ist zu dem geworden, was sie nun schließlich geworden ist – ein fantastisches Projekt mit unglaublichen Erfahrungen!

10.2. Ausblick

In Anbetracht der erzielten Fortschritte und Erkenntnisse bietet dieser Abschnitt einen Ausblick auf potenzielle zukünftige Entwicklungen und Implementierungsmöglichkeiten, um das untersuchte Thema weiter zu vertiefen und neue Perspektiven zu eröffnen.

Auf Seiten des *Frontends* gibt es vor allem bei dem Graphen noch wunderbare Ideen und Erweiterungen. Das bereits im Rechercheteil erwähnte Konzept des *Clusterings* ist nämlich mittels Bibliothek *Cytoscape* bereits möglich und könnte für Siemens eine äußerst nützliche und hilfreiche Anwendung finden. Die Struktur der Daten erlaubt diese Implementierung ebenfalls, da sie eine Art Baumstruktur mit verschachtelten Containern aufweist.

Außerdem könnte der interessante *Fisheye*-Effekt das Anzeigen von mehr Objekten im Graphen ermöglichen, da dem Benutzer die Fähigkeit des intuitiven „Heranzoomens“ gegeben wird. Eine derartige Arbeit könnte die Frage der Verbesserung der Effizienz bei Analyse- und Entscheidungsprozessen wortwörtlich unter die Lupe nehmen.

Abschließend lässt sich noch die Idee in den Raum werfen, zusätzliche relevante Informationen kompakt auf dem *Dashboard* anzuzeigen. Beispielsweise könnte eine kleine Version des Graphen mit Startelement eines schwerwiegenden Fehlers dort Platz finden.

I. Literaturverzeichnis

- [1] Papazoglou, Mike P. und Willem Jan Van Den Heuvel: *Service Oriented Architectures: Approaches, Technologies and Research Issues*. The VLDB Journal, 16(3):389–415, Juli 2007, ISSN 1066-8888, 0949-877X.
- [2] Toshev, Martin: *Learning RabbitMQ: Build and Optimize Efficient Messaging Applications with Ease*. Packt Publishing, Birmingham, 2016, ISBN 978-1-78398-456-5.
- [3] Curry, Edward: *Message-Oriented Middleware*. In: Mahmoud, Qusay H. (Herausgeber): *Middleware for Communications*, Seiten 1–28. Wiley, 1. Auflage, Juni 2004, ISBN 978-0-470-86206-3 978-0-470-86208-7.
- [4] Menge, Falko: *Enterprise Service Bus*. In: *Free and Open Source Software Conference*, Band 2, Seiten 1–6, 2007.
- [5] Fu, Guo, Yanfeng Zhang und Ge Yu: *A Fair Comparison of Message Queuing Systems*. IEEE Access, 9:421–432, 2021, ISSN 2169-3536.
- [6] Narkhede, Neha: *Kafka: The Definitive Guide*. O'Reilly Media, 2017.
- [7] Stopford, Ben: *Designing Event-Driven Systems*. O'Reilly Media, Inc., 2018.
- [8] Ohlbach, Hans Jürgen: *Graphen*. Ludwig-Maximilians-Universität München, 2018.
- [9] Klein, Felix: *Zusammenhang von Graphen*. Mathepedia.
- [10] Läuchli, Peter und Peter Läuchli: *Planarität*. Algorithmische Graphentheorie, Seiten 83–98, 1991.
- [11] Schmit, Anne Marie: *Der Satz von Kuratowski*. Paris-Lodron-Universität Salzburg, 2018.
- [12] Brandstädt, Andreas und Andreas Brandstädt: *Eulerkreise und Hamiltonkreise*. Graphen und Algorithmen, Seiten 40–60, 1994.
- [13] Liebling, Thomas M und Thomas M Liebling: *Euler-Graphen,-Zyklen und-Kreise*. Graphentheorie in Planungs-und Tourenproblemen: am Beispiel des städtischen Straßendienstes, Seiten 24–31, 1970.
- [14] Reith, Steffen und Heribert Vollmer: *Ist $P = NP$? Einführung in die Theorie der NP-Vollständigkeit*. Technischer Bericht, Technical Report 269, Institut für Informatik, Universität Würzburg, 2001 ..., 2001.
- [15] Prof. Dr. Glinz, Martin: *Informatik II: Modellierung*. Universität Zürich - Institut für Informatik, 2005.
- [16] Pfefferer, Leo: *Objektzentrierte Visualisierung mehrdimensionaler Daten als Erweiterung konventioneller Datenbankmodelle*. Herbert Utz Verlag, 1996.
- [17] Richter, Michael: *Kriterien der Benutzerfreundlichkeit*. Psychologisches Institut der Universität Zürich, November 1997.
- [18] Krug, Steve: *Don't make me think!: Web & Mobile Usability: Das intuitive Web*. MITP-Verlags GmbH & Co. KG, 2018.
- [19] Mardita, Rizki: *Designing Intuitive User Interface*. Medium, Mai 2017.
- [20] Vassilatos, Fanny und Ceara Crawshaw: *Enterprise Filtering - UX Pattern Analysis*. Pencil & Paper, 2023.
- [21] Hahn, Martin: *Icons für deine Website: So nutzt du sie richtig*. Webdesign Journal, 2024.

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

- [22] Laubheimer, Page: *The 3-Click Rule for Navigation Is False*. Nielsen Norman Group, 2019.
- [23] Wright, Chris: *Web Optimization: The Myth of the 3 Click Rule*. CMSWire, 2010.
- [24] Ediger, Natalie: *What is Interactive Content?* cleverclip, 2020.
- [25] Neuburger, Christoph: *Interaktivität, Interaktion, Internet*. Publizistik, 52(1):33–50, 2007.
- [26] Morville, Peter und Jeffery Callender: *Search patterns: design for discovery*. O'Reilly Media, Inc., 2010.
- [27] Shneiderman, Ben: *The eyes have it: A task by data type taxonomy for information visualizations*. In: *Proceedings 1996 IEEE symposium on visual languages*, Seiten 336–343. IEEE, 1996.
- [28] Ahlberg, Christopher, Christopher Williamson und Ben Shneiderman: *Dynamic queries for information exploration: An implementation and evaluation*. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, Seiten 619–626, 1992.
- [29] Souders, Steve: *High-performance web sites*. Communications of the ACM, 51(12):36–41, 2008.
- [30] Sharma, Sushil und Pietro Murano: *A usability evaluation of Web user interface scrolling types*. First Monday, 2020.
- [31] Hogan, Lara Callender: *Designing for Performance: Weighing Aesthetics and Speed*. O'Reilly Media, Inc., 2014.
- [32] Jorgensen, Corinne und Elizabeth D Liddy: *Information access or information anxiety?—An exploratory evaluation of book index*. Indexer, 1(20):64–68, 1996.
- [33] Barnum, Carol, Earvin Henderson, Al Hood und Rodney Jordan: *Index versus full-text search: a usability study of user preference and performance*. Technical Communication, 51(2):185–206, 2004.
- [34] statcounter.
- [35] Fry, Ben: *Visualizing data*. O'Reilly Media, Inc., 2008.
- [36] Cheng, Jeffrey: *The Top 10 Types of Data Visualization Made Simple*. Boost Labs, 2020.
- [37] Morales, José Miguel: *What is Data Visualization? From Data to Visualization*. BI-UWER, 2020.
- [38] French, Katie: *25 Tips to Instantly Improve Your Data Visualization Design*. Column Five, 2021.
- [39] insightsoftware: *Welches Diagramm für welche Daten? So finden Sie die richtige Diagrammtypen*. insightsoftware, 2023.
- [40] Marktler, Josef: *12 coole Typen – Wer ist der Richtige?* storytellingmitdaten, 2020.
- [41] Zeising, Tobias: *Was lesen Buchblogger: Eine neue Analyse mit Visualisierungen und Statistiken*. lesestunden, 2016.
- [42] Lin, Pohan: *2023 Guide to Big Data Visualization*. Piktochart, 2023.

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT
	Krems Abteilung: Informationstechnologie

- [43] Heins, Eberhard: *Daten-Tools visualisieren Big-Data-Potenzial*. the IT-Matchmaker, 2017.
- [44] Wampfler, Philippe: *Methodische Probleme von Big Data im Umgang mit Social Media*, 2014.
- [45] Pusic, Ante: *Graph Clustering Algorithms: Usage and Comparison*. memgraph, 2023.
- [46] Yi, Mike: *A complete guide to heatmaps*. Atlassian, 2020.
- [47] Cinar, Özlem: *Visualisierung der Ergebnisse (MOS) Heatmap*. CapeReviso, 2023.
- [48] Kassamara, Alboukadel: *Hierarchical Clustering in R: The Essentials Heatmap in R: Static and Interactive Visualization*. DataNovia, 2020.
- [49] Bostock, Mike: *Fisheye Distortion*. Mike Bostock, 2012.
- [50] Beilhammer, Marius: *Datenvisualisierung: Dateninterpretation auf einen Blick*. industryPRESS, 2017.
- [51] Jünger, Michael, Petra Mutzel, Stephen Kobourov, Sue Whitesides, Andreas Kerren, Holger Eichelberger, Martin Harrigan, Patrick Healy und Michael Belling: *Graph Drawing*. In: *Proc. Dagstuhl Seminar [online]*, Band 5191. Citeseer, 2002.
- [52] Mutzel, Petra, Michael Jünger und Sebastian Leipert: *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23-26, 2001, Revised Papers*, Band 2265. Springer, 2003.
- [53] Dokumentation von Angular Version 17 (modernisiert).
- [54] Ahmed, Nisar: *What Are Standalone Components and How to Utilize Them in Angular?* Medium, 2023.

II. Abbildungsverzeichnis

2.1.	Diese Darstellung zeigt den schematischen Aufbau der Applikation.	12
2.2.	This illustration shows the architecture of our application.	13
3.1.	Darstellung eines Systems in der <i>Service-Oriented Architecture</i>	16
3.2.	Ein Beispiel für ein verteiltes System mit direkter Kommunikation [3]	18
3.3.	System mit einer Datenbank als zentrale Kommunikationskomponente . . .	19
3.4.	Messaging System als zentraler Kommunikationsknoten [2]	20
3.5.	Ein Beispiel für ein verteiltes System mit ein zentrales <i>Messaging System</i> . .	21
3.6.	Ein einfacher <i>Enterprise Service Bus</i>	22
3.7.	Eine <i>Queue</i> [3]	23
3.8.	Das <i>Point-to-Point Messaging Model</i> [3]	25
3.9.	Das <i>Publish-Subscribe Messaging Model</i> [3]	25
3.10.	Die Architektur des Apache Kafka Systems [5]	30
3.11.	Eine <i>Consumer-Group</i> verarbeitet gemeinsam ein Topic. [6]	31
3.12.	Die Architektur des RabbitMQ Systems [5]	32
4.1.	Zwei Graphen dessen Unterschied die Ausrichtung ist	34
4.2.	Ein Graph all dessen Knoten verbunden sind (links); ein Graph, welcher nicht verbundene Knoten enthält (rechts)	35
4.3.	Ein Graph, welcher keinen geschlossenen Kreis (Zyklus) enthält (links); ein Graph mit Zyklus (rechts)	36
4.4.	Ein Graph, welcher gewichtete Kanten hat (links); ein Graph, dessen Kanten immer das gleiche Gewicht haben (rechts)	36
4.5.	Ein Graph, wo jeder Knoten mit allen anderen Knoten verbunden ist (links); nicht alle Knoten sind miteinander verbunden (rechts)	37
4.6.	Graph, welcher zwei Mengen ohne inhärente Verbindungen hat (links); keine Bipartition möglich beim rechten Graph	37
4.7.	Ein Graph ohne Schnittpunkte der Kanten, wenn alle Knoten verbunden sind (links); nicht möglich ohne Schnittpunkte (rechts)	38
4.8.	Graph mit einem Zyklus, welcher alle Kanten einmal enthält (links); Graph ohne diesen Eulerkreis	38
4.9.	Graph mit Zyklus, welcher alle Knoten beinhaltet (links); kein Hamiltonscher Kreis möglich (rechts)	39
5.1.	Einfache Datenvisualisierungen in Form von Werten, Diagrammen, Tabellen und Karten [37]	43
5.2.	Beliebtheit der verschiedenen Buchgenre im Jahr 2016 [41]	45
5.3.	Stark vernetzte Daten in einem Graphen (LinkedIn Netzwerk) [44]	46
5.4.	Clustering eines Graphen [45]	47
5.5.	Heatmap der Beiträge eines GitHub Benutzers im Jahr 2022 [46]	48
5.6.	Demonstration der Lupentechnik Fisheye auf einem Grid [49]	48
5.7.	Demonstration des Fischauges in einem Graphen [Source: GitHub Gist] . . .	49
5.8.	Demonstration der Lupentechnik Kartesisch auf einem Grid [49]	49
5.9.	Exemplarische geobasierte Heatmap aus der Google Maps Dokumentation .	50
5.10.	Architektur unseres Prototypen mit Fokus auf die Bibliothek Cytoscape . . .	51
6.1.	Übersicht der Architektur unseres Prototypen	52

6.2.	Architektur unseres Prototyps mit Fokus auf Kafka	53
6.3.	Architektur unseres Prototyps mit Fokus auf das <i>Backend</i>	54
6.4.	Architektur unseres Prototypen mit Fokus auf das <i>Frontend</i>	57
6.5.	Speicherung von Daten unseres Beispieles in einem Graphen	58
6.6.	Auflistung der Abfragen, welche mittels <i>GraphQL</i> implementiert sind [Schema Definition]	60
6.7.	Architektur unseres Prototypen mit Fokus auf das Frontend, welches mit Angular implementiert ist	60
6.8.	Abbildung der Tabelle der Fehlerdaten im Stromnetzwerk mit Sortier- und Filteroptionen unseres Prototypen	62
7.1.	Tabellenstruktur des Prototyps	69
8.1.	Sneak Peak unserer Angular Applikation	70
8.2.	Donatdiagramm	70
8.3.	Dashboard des Prototypen	71
8.4.	Interaktion mit den Diagrammen des Dashboards	72
8.5.	Tabelle mit den aktuellen Fehlern im Stromnetzwerk	73
8.6.	Sortierungsoptionen der Tabelle des Prototypen	73
8.7.	Filterungsoption mit Eingabefeld über alle Spalten	74
8.8.	Filterungsmöglichkeit mit Dropdown-Menu für Schweregrad	74
8.9.	Paginierung der Tabelle mit den Fehlern	75
8.10.	Einträge, welche über einen Spannungswert von 220.0kV verfügen	76
8.11.	Visualisierung eines Teils des Stromnetzwerkes in einem Graphen	77
8.12.	Aktuelles fehlerhaftes Netzmodellobjekt im Graphen blau markiert	78
8.13.	Abwechselnde Vorkommnis der verschiedenen Elementgruppen im Stromnetzwerk	78
8.14.	<i>Dashboard</i> im <i>Dark Mode</i>	79
8.15.	Ergebnisse des Netzmodells im <i>Dark Mode</i>	80
8.16.	Visualisierung des Stromnetzmodells mittels Graph im <i>Dark Mode</i>	80
A.1.	Jira Plan	118

III. Tabellenverzeichnis

A.1. Kapitelverzeichnis	92
A.2. Arbeitstagebuch Schlipfinger	120
A.3. Arbeitstagebuch Schneider	124

IV. Quellcodeverzeichnis

6.1.	eine einfache <i>GraphQL</i> -Abfrage	58
6.2.	das Ergebnis einer einfachen <i>GraphQL</i> -Abfrage	59
6.3.	Erstellen der Knoten und Kanten mittels Liste von Objekten von <i>GraphQL</i> .	62
6.4.	Laden der Knoten und Kanten in den Graphen mittels <i>Cytoscape</i>	63
7.1.	Docker Compose File zur einfachen Einrichtung von <i>Apache Kafka</i>	65
7.2.	Auszug aus der Konfigurationsdatei für <i>Spring</i>	66
7.3.	Definiton eines Topics mithilfe von <i>Kafka for Spring Framework</i>	66
7.4.	Definiton eines Consumers mithilfe von <i>Kafka for Spring Framework</i>	66
A.1.	Applikation ausführen mittels Docker Compose	125
A.2.	Testdaten in PostgreSQL einspielen	125

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT
	Krems
	Abteilung: Informationstechnologie

V. Abkürzungsverzeichnis

API Application Programming Interface

CRUD Create - Read/Retrieve - Update - Delete/Destroy

DI Dependency Injection

DNS Domain Name System

HCI Human Computer Interaction

HTTP Hypertext Transfer Protocol

SOA Service-Oriented Architecture

UI User Interface

UX User Experience

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

VI. Übersetzungsverzeichnis

Dashboard Grafische Übersicht, Schnelle Einblicke; Kontext: Stromnetzmodell

Framework Rahmenwerk - Entwicklungsgerüst, vorgefertigte Struktur, abstrahierte Umgebung

Library Bibliothek - Sammlung von Funktionen, wiederverwendbare Code-Module

Component Komponente - Teil eines Softwarepakets; Teil einer Applikation

A. Anhang

A.1. Kapitelverzeichnis

Kapitel	Autor
1. Präambel	Schneider Felix und Schlipfinger Clemens
2. Einleitung	Schneider Felix und Schlipfinger Clemens
3. Message Propagation	Schlipfinger Clemens
4. Graphentheorie	Schneider Felix
5. Visualisierung	Schneider Felix
6. Architektur des Prototypen	Schneider Felix
6.1 Apache Kafka und das Backend	Schlipfinger Clemens
6.2 GraphQL und das Frontend	Schneider Felix
7. Das Backend-System	Schlipfinger Clemens
8. Schnittstelle GraphQL	Schlipfinger Clemens
9. Webapplikation Angular	Schneider Felix
10. Bewertung des Enterprise Service Bus Systems	Schlipfinger Clemens
11. Bewertung der Visualisierungsmethoden im Frontend	Schneider Felix
12. Zusammenfassung	Schneider Felix und Schlipfinger Clemens
13. Anhang	Schneider Felix

Tabelle A.1.: Kapitelverzeichnis

A.2. Besprechungsprotokolle

A.2.1. Begleitprotokoll 1



Begleitprotokoll 1

Meeting Minutes

—

Project

Id 202324-CF-Networkanalysis

Name Visualisierung der Ergebnisse der Netzdatenmodellanalyse

Team

Jürgen Katzenschlager Project manager

Clemens Schlipfänger Backend programmer

Felix Schneider Frontend programmer

Version History

Version	Datum	AutorIn	Änderungen
0.1	28.10.2023	Felix Schneider & Clemens Schlipfänger	Erstellung des Dokuments



Inhaltsverzeichnis

[Table of Contents]

Inhaltsverzeichnis.....	1
Metadaten.....	2
Ergebnisse.....	2
Inhalte der Besprechung.....	2
Themen.....	2
Next Steps.....	2

Metadaten

- Datum: 28.10.2023
- Zeit: 18:00 - 19:02 Uhr
- Ort: Home Office
- Anwesenden Personen
 - Jürgen Katzenschlager
 - Clemens Schlipfinger
 - Felix Schneider

Ergebnisse

- Pflichtenheft erstellt und Siemens geschickt
- Mockup halb fertig

Inhalte der Besprechung

Themen

- Pflichtenheft
 - Datum bei Suche?
 - Deployment/Abgabe als Docker-Container?
 - Mockup:
 - Dashboard mehr Content
 - Accounts müssen weg!
 - Filter: Sortierung
 - Zeichnung der Systemarchitektur mit Farben und Icons erweitern.

Next Steps

- Jira Plans
 - Arbeitspakete definieren
- Mockup und Zeichnung der Systemarchitektur verfeinern
- Art der Abgabe herausfinden (Siemens fragen)

A.2.2. Begleitprotokoll 2



Begleitprotokoll 2

Meeting Minutes

—

Project

Id 202324-CF-Networkanalysis

Name Visualisierung der Ergebnisse der Netzdatenmodellanalyse

Team

Jürgen Katzenschlager Project manager

Clemens Schlipfing Backend programmer

Felix Schneider Frontend programmer

Version History

Version	Datum	AutorIn	Änderungen
0.1	13.12.2023	Felix Schneider & Clemens Schlipfing	Erstellung des Dokuments



Inhaltsverzeichnis

[Table of Contents]

Inhaltsverzeichnis.....	1
Metadaten.....	2
Ergebnisse.....	2
Inhalte der Besprechung.....	2
Themen.....	2
Next Steps.....	2

Metadaten

- Datum: 13.12.2023
- Zeit: 11:13 - 11:46 Uhr
- Ort: 5AHIT
- Anwesenden Personen
 - Jürgen Katzenschlager
 - Clemens Schlipfinger
 - Felix Schneider

Ergebnisse

- Datenbankmodell fertig

Inhalte der Besprechung

Themen

- Datenbankmodell
 - Equipments (Connected vs Connecting)
 - Single Table machen?
 - Node_has_equipment entfernen
 - Findings Types
- Strimzi: Kafka Topics Config

Next Steps

- Jira Plan exportieren als Bild
- Jira Issues aufräumen und Spring planen
 - Research und Theoretische Grundlagen
 - Definition der API
 - Kafka Konfigurieren (Topics, ...)
 - Database Schema fertig stellen
- Postman Collections anschauen, damit Frontend nicht abhängig

A.2.3. Begleitprotokoll 3



Begleitprotokoll 3

Meeting Minutes

—

Project

Id 202324-CF-Networkanalysis

Name Visualisierung der Ergebnisse der Netzdatenmodellanalyse

Team

Jürgen Katzenschlager Project manager

Clemens Schlipfinger Backend programmer

Felix Schneider Frontend programmer

Version History

Version	Datum	AutorIn	Änderungen
0.1	07.01.2023	Felix Schneider & Clemens Schlipfinger	Erstellung des Dokuments



Inhaltsverzeichnis

[Table of Contents]

Inhaltsverzeichnis.....	1
Metadaten.....	2
Ergebnisse.....	2
Inhalte der Besprechung.....	2
Themen.....	2
Next Steps.....	2

Metadaten

- Datum: 07.01.2023
- Zeit: 18:03 - 18:27 Uhr
- Ort: Home Office
- Anwesenden Personen
 - Jürgen Katzenschlager
 - Clemens Schlipfinger
 - Felix Schneider

Ergebnisse

- Datenbankmodell und API definiert
- Kafka mit Kubernetes rennt, Topics aufgesetzt
- Angular Dark/Light Mode integrieren
- Spring Framework informieren, PostgreSQL
- Theoretischer Teil mit Bildern von Felix fertig, Clemens angefangen

Inhalte der Besprechung

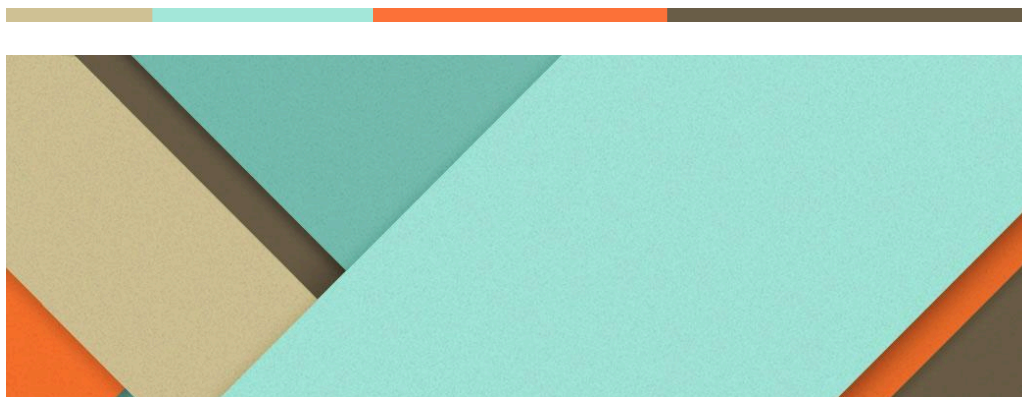
Themen

- Theoretischen Teil
- Jira Plan anschauen
 - Nicht erledigte Issues (mitten im Sprint hinzugefügt) zurück in Backlog
- Empirischen Teil
 - Zuerst Architektur beschreiben, dort kommen dann neue Themen, zB GraphQL, welche kurz beschrieben werden können
 - Interessante / schwierige Themen, welche während der Programmierung aufgetreten sind beschreiben (Notizen machen)

Next Steps

- Jira Sprint planen (Januar)
 - Prototypen (Felix und Clemens)
 - Theoretischen Teil fertig schreiben (Clemens)
- Abbildungen besser zum Thema / Überschrift beschreiben

A.2.4. Begleitprotokoll 4



Begleitprotokoll 4

Meeting Minutes

—

Project

Id 202324-CF-Networkanalysis

Name Visualisierung der Ergebnisse der Netzdatenmodellanalyse

Team

Jürgen Katzenschlager Project manager

Clemens Schlipfing Backend programmer

Felix Schneider Frontend programmer

Version History

Version	Datum	AutorIn	Änderungen
0.1	13.02.2024	Felix Schneider & Clemens Schlipfing	Erstellung des Dokuments; Meeting verschieben aufgrund Zeitmangel
0.2	16.02.2024	Felix Schneider & Clemens Schlipfing	Fertigstellung des Dokuments



Inhaltsverzeichnis

[Table of Contents]

Inhaltsverzeichnis.....	1
Metadaten.....	2
Ergebnisse.....	2
Inhalte der Besprechung.....	2
Themen.....	2
Next Steps.....	2

Metadaten

- Datum: 16.02.2023
- Zeit: 13:34 - 15:20 & 17:52 - 19:01 Uhr
- Ort: HTL Krems + Home Office
- Anwesenden Personen
 - Jürgen Katzenschlager
 - Clemens Schlipfinger [nur erstes Teilmeeting]
 - Felix Schneider

Ergebnisse

- Implementierung des Prototypen weit vorangeschritten
- Prof. Katzenschlager hat den theoretischen Teil verbessert (großes Dankeschön!)

Inhalte der Besprechung

Themen

- Schriftlicher Teil korrigiert (durchgehen und ausbessern)
 - Einige Rechtschreibfehler
 - Inhalt und Überschriften einleiten und erklären, worum es gehen wird; nicht einfach reinplatzen.
 - Inhaltlich UI vs. Datenanalyse (Felix)

Next Steps

- Implementierung abschließen
 - MobelObject API fertigstellen
 - Finding Table fertigstellen
 - Graph abschließen
- Theoretischen Teil verbessern
 - Clemens: Einleitung, deutsche Sätze, Struktur anpassen
 - Felix: Umstrukturierung (Graphentheorie eigenes Kapitel), Themen mehr auf stark vernetzte Daten spezialisieren
- Empirischen Teil anfangen

A.2.5. Begleitprotokoll 5



Begleitprotokoll 5

Meeting Minutes

—

Project

Id 202324-CF-Networkanalysis

Name Visualisierung der Ergebnisse der Netzdatenmodellanalyse

Team

Jürgen Katzenschlager Project manager

Clemens Schlipfing Backend programmer

Felix Schneider Frontend programmer

Version History

Version	Datum	AutorIn	Änderungen
0.1	24.03.2024	Felix Schneider & Clemens Schlipfing	Erstellung des Dokuments;



Inhaltsverzeichnis

[Table of Contents]

Inhaltsverzeichnis.....	1
Metadaten.....	2
Ergebnisse.....	2
Inhalte der Besprechung.....	2
Themen.....	2
Next Steps.....	2

Metadaten

- Datum: 24.03.2023
- Zeit: 20:54 - 21:39 Uhr
- Ort: Home Office
- Anwesenden Personen
 - Jürgen Katzenschlager
 - Clemens Schlipfinger
 - Felix Schneider

Ergebnisse

- Empirischer Teil fast fertig
- Viele kleine Verbesserungen
- Viele Fragen bzgl Abgabe und so

Inhalte der Besprechung

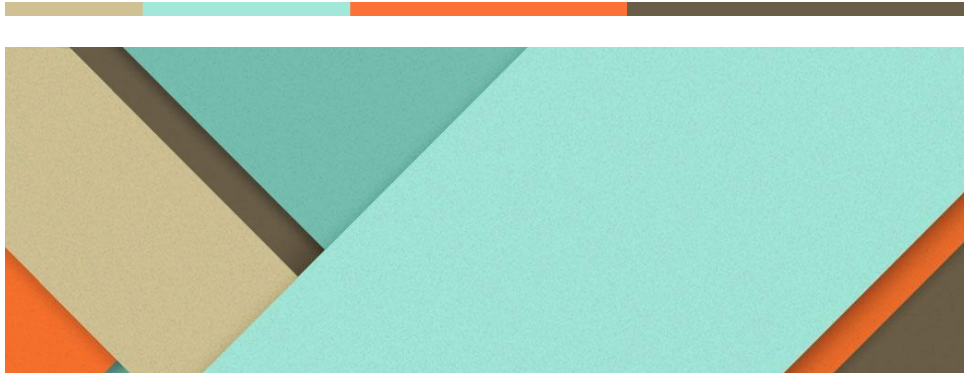
Themen

- Fragen beantworten (USB-Stick, Management-Dokumente, Quellen, Warning, ...)
- Empirischer Teil Inhalt nochmals detailliert
- Bewertung ein Kapitel reicht für beide
- Herr Professor einen guten Flug und einen schönen Urlaub wünschen!

Next Steps

- Clemens Empirischer Teil bis morgen 12:00 Uhr zum Drüberlesen (eigener Wunsch)
- Bewertung fertig schreiben

A.3. Pflichtenheft



Pflichtenheft

Requirements Specification

—

Project

Id 202324-CF-Networkanalysis

Name Visualisierung der Ergebnisse der Netzdatenmodellanalyse

Team

Jürgen Katzenschlager Project manager

Clemens Schlipfinger Backend programmer

Felix Schneider Frontend programmer

Version History

Version	Datum	AutorIn	Änderungen
0.1	17.08.2023	Felix Schneider	Erstellung des Dokuments
0.2	22.08.2023	Felix Schneider	Hinzufügen einiger Ziele
0.3	23.08.2023	Felix Schneider	Ziele schreiben [überarbeitungswürdig]
0.4	10.10.2023	Felix Schneider	Funktionale Anforderungen Grafik
0.5	11.10.2023	Clemens Schlipfinger	Systemarchitektur Grafik hinzufügen
0.5	25.10.2023	Felix Schneider	Mockup hinzufügen
0.6	25.10.2023	Felix Schneider	Nicht Funktionale Anforderungen anfangen
0.7	26.10.2023	Felix Schneider	Überarbeitung Ziele Frontend, technische Anforderungen
0.8	27.10.2023	Clemens Schlipfinger	Hinzufügen Backend Ziele und Anforderungen
0.9	30.10.2023	Felix & Clemens	Überarbeitung mit Siemens



Inhaltsverzeichnis

[Table of Contents]

Inhaltsverzeichnis.....	2
Ziele.....	3
MUSS-Ziele.....	3
KANN-Ziele.....	4
NICHT-Ziele.....	4
Funktionale Anforderungen.....	5
Anforderungen an das Backend.....	5
Optionale Ziele.....	5
Anforderungen des Frontends.....	6
Nicht funktionale Anforderungen.....	7
Technische Anforderungen.....	8
Übersicht.....	8
Backend.....	8
Frontend.....	8
Mockup.....	9

Ziele

[Goals]

MUSS-Ziele

Die Muss-Ziele definieren klar, welche Anforderungen unbedingt erfüllt sein müssen. Alle Ziele sind bis **12.04.2024** zu erfüllen.

- ☐ Backend
 - ☐ Der Backend-Server soll die Daten (Findings) persistieren und für die Webanwendung in einer sinnvollen Art zur Verfügung stellen.
 - ☐ Der Backend-Server soll in der Lage sein, gleichzeitig Daten von verschiedenen Netzmodellanalysen zu verarbeiten.
 - ☐ Der Backend-Server soll sich leicht in das bestehende System integrieren und als optionale Erweiterung des Systems gelten.
 - ☐ Der Backend-Server soll asynchron funktionieren, welches eine gleichzeitige Verarbeitung und Bereitstellung der Daten ermöglicht.
- ☐ Frontend
 - ☐ Die Webanwendung bietet umfangreiche Suchmöglichkeiten in tabellarischer Form, nach den Kriterien Schweregrad, Kategorie, Spannungsebene, ID des Findings und der technischen Adresse des Findings..
 - ☐ Die Webanwendung visualisiert Relationen im Stromnetzwerk mittels Graphen.
 - ☐ Die Webanwendung stellt Informationen übersichtlich in einem Dashboard dar.

KANN-Ziele

Die Kann-Ziele sind optional.

- ☐ Backend
 - ☐ Das Backend soll auch den Client über neue Daten informieren.
 - ☐ Die API verfügt über eine Authentifizierung mittels Siemens Integration.
- ☐ Frontend
 - ☐ Die Webanwendung stellt Zusammenhänge in Diagrammen dar.

NICHT-Ziele

Die Nicht-Ziele sind explizit nicht zu erfüllen. Um Verneinungen zu vermeiden, sind diese trotzdem so formuliert, als würden sie erfüllt werden müssen.

- ☐ Die Fehler des Netzmodells sollen erkannt und in die Findings (Java Objekt) gespeichert werden.
- ☐ Es wird im bestehenden Repository von Siemens gearbeitet.
- ☐ Die Applikation wird auch für mobile Geräte entwickelt.

Funktionale Anforderungen

[Functional Requirements]

Anforderungen an das Backend

Das Backend soll die Daten der Netzmodellfehleranalyse verarbeiten. Die Daten sind die Findings, welche die Ergebnisse der Netzmodellfehleranalyse sind.

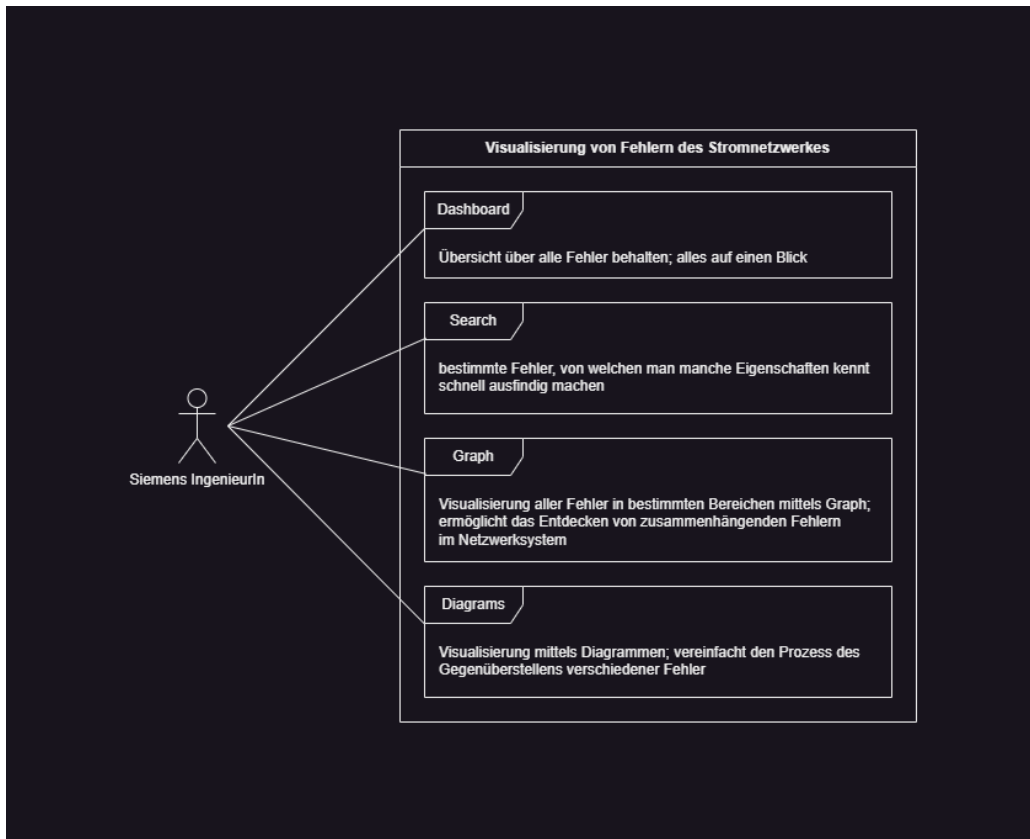
- Es müssen die Daten mit Hilfe einer Datenbank **persistiert** werden und bis zur nächsten Netzmodellfehleranalyse gehalten werden.
- Die Findings müssen durch eine **Schnittstelle (GraphQL) dem Frontend** angeboten werden.
- Der Backend soll sich als **optionale Erweiterung** leicht in das bisherige System integrieren lassen. Zusätzlich soll es nur kleine Veränderungen am Haupt-System erzwingen.
 - Für den Entwicklungsprozess soll eine **Simulationssoftware** verwendet werden.
 - Es kann davon ausgegangen werden, dass ein Kafka-System und eine Datenbank schon bestehen. Das Programm, welches die Daten in das Kafka Topic schreibt, wird von Siemens bereitgestellt.
- Das Backend soll **verschiedene Netzmodellfehleranalyse** verarbeiten können, das heißt, es muss Findings von mehreren Exportmodulen gleichzeitig entgegennehmen können.

Optionale Ziele

- Es kann eine Authorization für das Einschränken des Zugriffs auf die Schnittstelle zum Frontend geben.
- Der Backend-Server soll über die Funktionalität verfügen, den Client mit **Server-Side Push Updates** über den Eingang von neuen Daten zu informieren.

Anforderungen des Frontends

Dieses Use Case Diagramm veranschaulicht die Funktionalitäten des Frontends:



Nicht funktionale Anforderungen

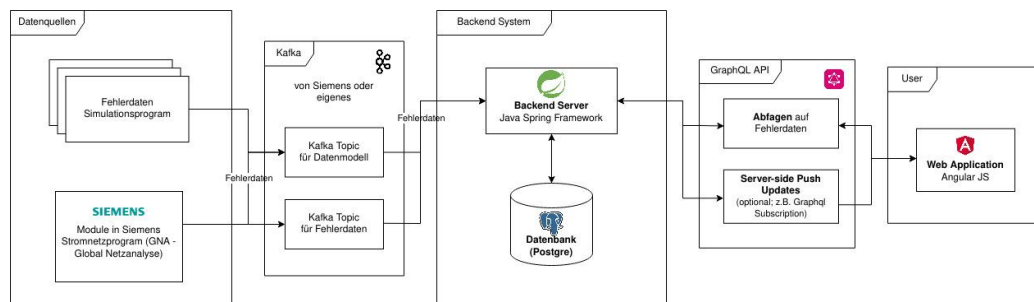
[Non-Functional Requirements]

Aspekt \ Projektergebnis	Webanwendung	Kafka-System
Usability	Intuitive Benutzbarkeit	-
Design	Nicht responsive für mobile Geräte	-
Navigation	Einfache Menüs, max. drei Klicks notwendig, um irgendwohin zu kommen	-
Look & Feel	Anlehnung an Siemens Applikationen; Intuitiv	Integration Siemens Kafka System
Barrierefreiheit	Kein TTS, Zoom oder Augensteuerung	-
Mehrsprachigkeit	Prinzipiell alles English, Icons ermöglichen eingeschränkte multilinguale Nutzung; Möglichkeit auf Erweiterung der Sprachen mittels Konfigurationsfile	-
Customisation	Dark / Light Mode	-
Zeitverhalten	-	Schnelle Änderungen

Technische Anforderungen

In diesem Kapitel wird der Technologie-Stack detaillierter beschrieben.

Übersicht



Backend

Die **Daten**, welche über das Kafka System übertragen werden können in **zwei Arten** unterteilt werden:

- Findings - Informationen über die gefundenen Fehler
- Datenmodell Objekte - Informationen über den Aufbau des Netzes

Diese zwei Arten von Daten werden über **zwei verschiedene Kafka Topics** erhalten.

Der Backend-Server, welcher mit dem **Java Spring Framework** implementiert ist, soll die Daten über die Event Streaming Plattform **Kafka** erhalten. Weiters muss es erhaltene Daten in einer **Datenbank** speichern und als **GraphQL API** den Frontend zur Verfügung stellen.

Frontend

Die Webanwendung verwendet das AngularJS Framework. Genau wie React, Vue und viele andere JS Frameworks basiert dieses auf Komponenten.

Über eine GraphQL API können Abfragen an das Backend gestellt werden.

Mockup

Das Mockup für die Web-Applikation wurde mit Figma erstellt. Hier ist ein Link zu der Datei:
<https://www.figma.com/file/PpiE28AJCmFWBdhD66wYX5/Siemens?type=design&mode=design&t=pvAVxSDRdFICOXGr-1>

Folgende Bilder veranschaulichen grob das Design:



figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

A.4. Arbeitspakete

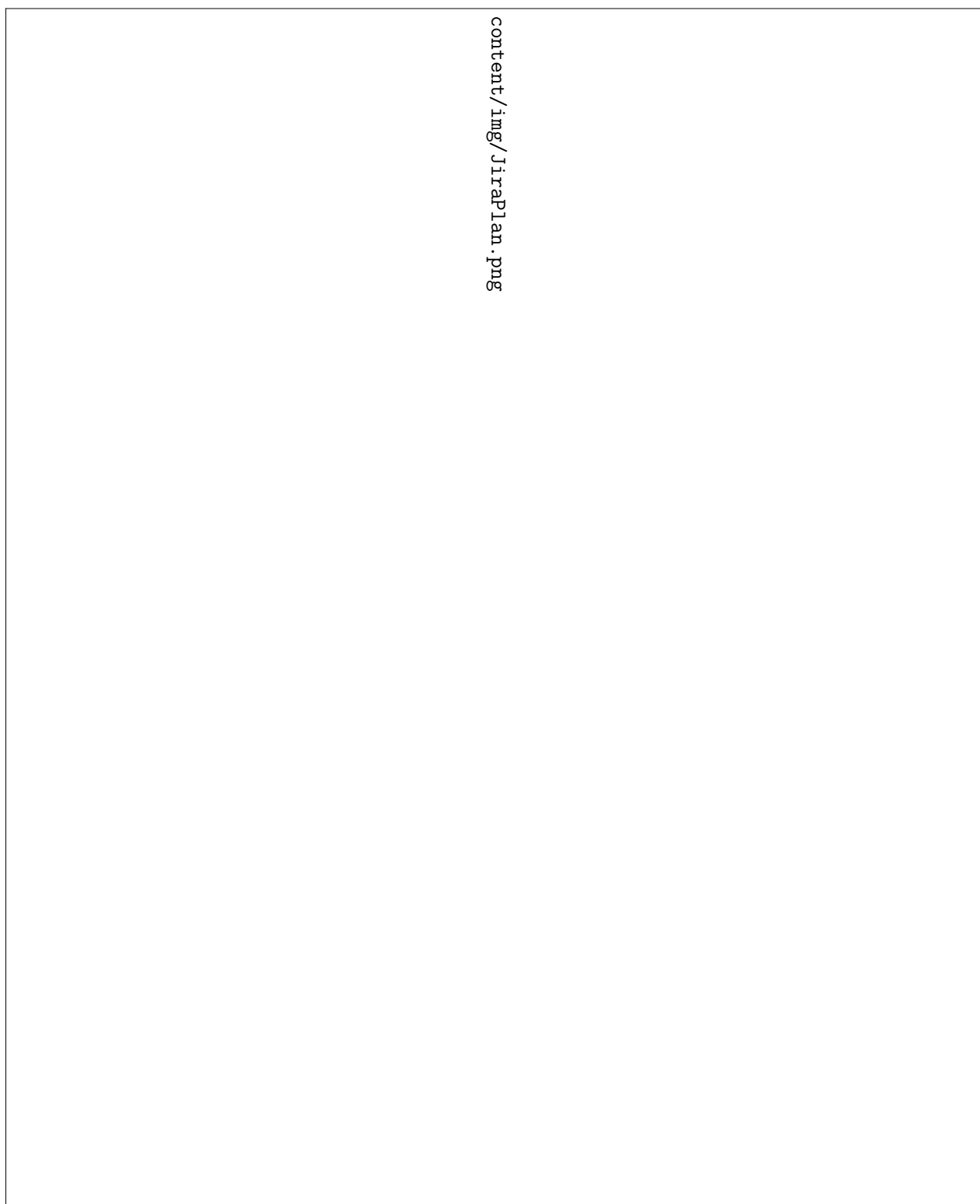


Abbildung A.1.: Jira Plan

A.5. Projekttagebücher

A.5.1. Projekttagebuch Clemens Schlipfing

Tag	Zeit	kumulativ	Fortschritt
11 October	00:29:32	00:29:32	Zeichnung der Systemarchitektur
27 October	01:04:08	01:33:40	Pflichtenheft
27 October	00:42:43	02:16:23	pflichtenheft
28 October	01:30:55	03:47:18	Meeting 1: Besprechung und Protokollierung
30 October	06:51:59	10:39:43	Siemens Meeting
31 October	01:58:31	12:38:14	Dokumentierung der Findings und ModellOb- jects
31 October	00:20:48	12:59:02	Analyse der ModellObjekte in einem Graphen
31 October	00:10:10	13:09:12	Analyse der ModellObjekte in einem Graphen
31 October	01:06:52	14:16:04	Analyse der ModellObjekte in einem Graphen
31 October	00:18:58	14:35:02	Analyse der ModellObjekte in einem Graphen
07 December	01:49:42	16:24:44	ewa, thoretische grundlagen
13 December	02:30:40	18:55:24	Databank Modell defininieren
13 December	00:29:43	19:25:07	Gespräch mit Begleiter
13 December	00:30:55	19:56:02	planning
18 December	01:23:12	21:19:14	GraphQL APi
18 December	01:17:30	22:36:44	GraphQL APi
18 December	00:54:43	23:31:27	GraphQL APi
20 December	01:49:50	25:21:17	graphql api
20 December	00:12:15	25:33:32	nan
20 December	00:23:58	25:57:30	GraphQL APi
23 December	00:29:21	26:26:51	research
23 December	04:30:00	30:56:51	research
27 December	00:22:27	31:19:18	literatur lesen
28 December	02:37:35	33:56:53	literatur lesen
28 December	01:30:39	35:27:32	theoretischen Teil
29 December	02:28:44	37:56:16	Kafka Topics einrichten / Meeting
29 December	00:58:24	38:54:40	kafka topics
29 December	01:53:01	40:47:41	kafka topics
30 December	05:29:53	46:17:34	Spring Boot
31 December	01:24:09	47:41:43	Spring Boot + Kafka
31 December	01:52:08	49:33:51	Spring Boot + Kafka
06 January	02:09:14	51:43:05	theoretischen Teil
07 January	02:06:34	53:49:39	Research
08 January	01:04:25	54:54:04	theoretischen Teil
08 January	00:40:39	55:34:58	theoretischen Teil
08 January	01:16:31	56:51:29	theoretischen Teil
09 January	02:07:44	58:59:13	theoretischen Teil
10 January	05:36:13	64:35:41	theoretischen Teil
11 January	00:55:26	65:31:07	theoretischen Teil
11 January	00:50:58	66:22:12	theoretischen Teil

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT		
	Krems		
	Abteilung:	Informationstechnologie	

11 January	01:21:51	67:44:03	theoretischen Teil
15 January	01:50:50	69:34:53	theoretischen Teil
16 January	02:05:31	71:40:24	theoretischer Teil
18 January	01:25:01	73:05:25	graphql
18 January	02:10:39	75:16:04	graphql
29 January	02:09:51	77:25:55	spring framework
30 January	01:53:36	79:19:31	spring framework
31 January	01:49:34	81:09:05	Hibernate
01 February	01:20:26	82:29:31	hibernate
01 February	02:56:46	85:26:17	hibernate
02 February	00:19:45	85:46:02	json
02 February	01:23:33	87:09:35	nan
03 February	00:53:10	88:02:45	backend
05 February	02:56:13	90:58:58	backend
06 February	01:03:41	92:02:39	JSON MAPPER
06 February	06:11:52	98:14:31	inserts in database
07 February	07:00:55	105:15:26	refactoring backend
08 February	03:29:31	108:44:57	Spring Data JDBC
08 February	02:10:06	110:55:03	Spring Data JDBC
09 February	04:29:32	115:24:35	writing queries
10 February	04:53:18	120:17:53	spring graphql
11 February	03:46:19	124:04:12	creating the findings endpoint
12 February	02:30:31	126:34:43	docker and bug fixing
12 February	02:14:17	128:49:00	bugfixes
12 February	00:48:28	129:37:28	container endpoint and modelobjects endpoint
15 February	01:25:29	131:02:57	improve findings
16 February	01:30:33	132:33:30	DA Meeting
17 February	05:56:40	138:30:10	modelobjects endpoint and filters
18 February	01:30:17	140:00:27	Graph Query Finished
20 February	01:25:07	141:25:34	kafka kafka kafka
20 February	01:20:06	142:45:40	kafka kafka
21 February	04:29:56	147:15:36	kafka kafka
06 March	01:13:22	148:28:58	Kafka and Dashboard; Backend
07 March	01:10:57	149:39:55	Kafka and Dashboard; Backend
11 March	01:10:40	150:50:35	Kafka and Dashboard; Backend
11 March	02:25:06	153:15:49	Kafka and Dashboard; Backend
13 March	03:29:49	156:45:38	Schriftlicher Teil; Theoretische Grundlagen
18 March	03:16:43	160:02:21	WKO Presentation
18 March	03:44:20	163:46:41	Schriftlicher Teil; Theoretische Grundlagen
20 March	01:12:06	165:03:11	Schriftlicher Teil; Theoretische Grundlagen
20 March	04:17:39	169:20:50	Schriftlicher Teil; Theoretische Grundlagen
20 March	02:20:32	171:41:22	Schriftlicher Teil; Theoretische Grundlagen
21 March	04:11:10	175:52:32	Schriftlicher Teil; Theoretische Grundlagen

Tabelle A.2.: Arbeitstagebuch Schlipfinger

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT
	Krems
	Abteilung: Informationstechnologie

A.5.2. Projekttagbuch Felix Schneider

Tag	Zeit	kumulativ	Fortschritt
14 September	00:08:33	00:08:33	Teams Team organisieren
18 September	00:12:06	00:20:39	Diplomarbeit Datenbank Felder ausfüllen
18 September	00:25:01	00:45:40	Clemens EWA erklären
18 September	00:07:24	00:53:04	Diplomarbeit Datenbank Felder ausfüllen
18 September	00:06:03	00:59:14	Diplomarbeit Datenbank Felder ausfüllen
19 September	00:36:28	01:35:52	Diplomarbeit Datenbank Felder ausfüllen
20 September	02:35:23	04:11:15	Mockup Figma View erstellen
21 September	00:57:12	05:08:27	Mockup Figma View erstellen
21 September	00:32:46	05:41:13	Mockup Figma View erstellen
21 September	01:43:46	07:24:59	Mockup Figma View erstellen
23 September	02:29:19	09:54:18	Diplomarbeit Datenbank Felder ausfüllen
10 October	00:18:50	10:13:08	UseCase Frontend für Plichtenheft
13 October	03:07:26	13:20:34	Research Graph Visualisation
19 October	01:52:29	15:13:03	Mockup
19 October	01:20:57	16:34:00	Mockup
19 October	01:03:47	17:37:47	Mockup
23 October	00:17:46	17:55:33	Mockup
23 October	00:20:25	18:15:58	Mockup
24 October	00:37:49	18:53:47	Mockup
25 October	01:27:02	20:22:53	Mockup
25 October	00:53:18	21:16:11	Mockup
25 October	00:27:04	21:43:15	Pflichtenheft
25 October	00:10:23	21:53:38	Mockup
26 October	00:35:09	22:29:59	Pflichtenheft
26 October	00:05:34	22:35:36	Mockup
27 October	00:51:59	23:27:35	Pflichtenheft
28 October	01:20:43	24:48:18	Besprechung
29 October	00:30:12	25:18:30	Mockup
29 October	00:23:14	25:41:44	Mockup
30 October	06:03:42	31:45:26	Meeting bei Siemens; Überarbeitung Pflichtenheft; Kleinigkeiten klären
01 November	00:29:38	32:15:04	Jira Plan erstellen
01 November	00:30:50	32:45:54	Jira Plan erstellen
01 November	00:29:53	33:15:47	Confluence weiterarbeiten
03 November	00:44:08	33:59:55	Jira Plan und Confluence
03 November	00:21:42	34:21:37	Mockup
03 November	00:31:16	34:52:53	Mockup
03 November	02:06:28	36:59:21	Learn Angular
16 November	00:50:59	37:50:20	Jira Plan erstellen
16 November	00:21:11	38:11:31	Jira Plan erstellen
10 December	00:20:38	38:32:09	Angular Table
10 December	00:51:16	39:23:25	Grundstruktur aufbauen
13 December	00:44:58	40:08:23	Meeting Minutes 2

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT		
	Krems		
	Abteilung:	Informationstechnologie	

13 December	03:02:06	43:10:29	Datenbankstruktur Modellierung und Research Visualisierungsmethoden
14 December	00:28:10	43:38:39	Research zitieren Benutzerfreundlichkeit
16 December	01:11:49	44:50:28	Research zitieren Benutzerfreundlichkeit
17 December	01:33:08	46:23:36	Basics herrichten Overleaf schriftliche Arbeit
17 December	01:46:57	48:10:33	Benutzerfreundlichkeit Recherche Bibs & Text bei Intuitivität und Suchmöglichkeiten
17 December	01:12:35	49:23:08	Benutzerfreundlichkeit Recherche Bibs & Text bei Suchmöglichkeiten und Performance
17 December	01:09:23	50:32:31	Verbesserungen der bestehenden Texte
18 December	01:01:34	51:34:05	API definieren
18 December	00:30:26	52:04:31	API definieren bei Spaziergang
18 December	01:17:36	53:22:07	API definieren
19 December	00:15:49	53:37:56	Verbesserung des Absatzes mit Three Click Method
19 December	01:37:44	55:15:40	Arten von Datendarstellungen
19 December	00:47:45	56:03:25	Arten von Datendarstellungen
19 December	00:29:37	56:33:02	Arten von Datendarstellungen
19 December	00:21:23	56:54:25	Arten von Datendarstellungen
19 December	01:03:59	57:58:24	Arten von Datendarstellungen Figma Designs
20 December	00:18:11	58:16:35	Arten von Datendarstellungen (verfeinern)
20 December	01:03:20	59:19:55	Arten von Datendarstellungen
20 December	00:15:27	59:35:22	Überarbeitung der Texte
21 December	01:04:48	60:40:10	Überarbeitung der Texte
21 December	00:12:11	60:52:21	Review GraphQL Schema
25 December	01:56:20	62:48:41	Angular Dark Light Mode
25 December	00:34:56	63:23:37	Angular Dark Light Mode
27 December	01:15:02	64:38:39	Libraries D3 & Cytoscape
27 December	00:45:33	65:24:12	Libraries D3 & Cytoscape
27 December	00:33:56	65:58:08	Libraries D3 & Cytoscape
01 January	02:44:55	68:43:03	Tabellen und Diagramme schreiben
06 January	01:14:27	69:57:30	Einleitung schreiben
06 January	00:44:14	70:41:44	Einleitung schreiben
07 January	00:48:21	71:30:05	LaTeX: Organisieren des schriftlichen Dokuments, Ordner umbenennen, aufräumen
07 January	02:07:23	73:37:28	LaTeX: Organisieren des schriftlichen Dokuments, Ordner umbenennen, aufräumen
07 January	01:22:07	74:59:35	Tabellen und Diagramme schreiben
18 January	00:31:47	75:31:22	Look over Theorie Part
18 January	01:41:36	77:12:58	Cytoscape in Angular integrieren
23 January	01:14:42	78:27:40	Cytoscape herumspielen, Animations, Layout, ...
24 January	00:30:04	78:57:44	Statusmeldung Bericht mit Siemens
24 January	01:01:10	79:58:54	Empirischer Teil schreiben, Graph
25 January	01:31:23	81:30:17	Angular Types erstellen (GraphQL Schema)
27 January	00:43:26	82:13:43	Look over Theorie Part

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT		
	Krems		
	Abteilung:	Informationstechnologie	

28 January	00:39:49	82:53:32	Angular Tabelle mit Such und Filteroptionen wie in Figma erstellen
28 January	01:34:19	84:27:51	Load Part of Graph (Model from Siemens)
30 January	00:59:37	85:27:28	Angular Tabelle mit Such und Filteroptionen wie in Figma erstellen
31 January	01:05:08	86:32:36	Angular Tabelle mit Such und Filteroptionen wie in Figma erstellen
31 January	01:13:30	87:46:06	Angular Tabelle mit Such und Filteroptionen wie in Figma erstellen
03 February	00:47:27	88:33:33	Dashboard Percentage
03 February	00:47:38	89:21:11	Load Part of Graph (Model from Siemens)
04 February	00:56:17	90:17:28	Load Part of Graph (Model from Siemens)
05 February	02:01:17	92:18:45	Load Part of Graph (Model from Siemens)
05 February	02:40:21	94:59:06	Load Part of Graph (Model from Siemens)
05 February	01:34:43	96:33:49	Load Part of Graph (Model from Siemens)
06 February	05:08:59	101:42:48	Load Part of Graph (Model from Siemens)
07 February	01:02:03	102:44:51	Load Part of Graph (Model from Siemens)
07 February	01:25:59	104:10:50	Empirischen Teil schreiben
10 February	00:38:05	104:48:55	Angular Tabelle mit Such und Filteroptionen wie in Figma erstellen
10 February	02:16:10	107:05:05	Angular Tabelle mit Such und Filteroptionen wie in Figma erstellen
10 February	03:16:16	110:21:21	Angular Tabelle mit Such und Filteroptionen wie in Figma erstellen
11 February	02:14:55	112:36:16	Angular Tabelle mit Such und Filteroptionen wie in Figma erstellen
13 February	02:23:17	114:59:33	Angular Tabelle mit Such und Filteroptionen wie in Figma erstellen
14 February	00:44:02	115:43:35	API Schema definieren
16 February	01:43:30	117:27:05	Theoretischen Teil nachbesprechen
16 February	01:30:49	118:57:54	Theoretischen Teil nachbesprechen
16 February	02:01:21	120:59:15	Theoretischen Teil überarbeiten
17 February	01:24:05	122:23:20	Angular Tabelle mit Such und Filteroptionen wie in Figma erstellen
17 February	04:59:33	127:22:53	Angular Tabelle mit Such und Filteroptionen wie in Figma erstellen
18 February	01:36:34	128:59:27	Theoretischen Teil überarbeiten
19 February	01:44:59	130:44:26	Update Angular new Schema
19 February	03:10:07	133:54:33	Angular Graph and Diagrams
20 February	00:42:52	134:37:25	Angular Graph and Diagrams
20 February	02:49:01	137:26:26	Angular Graph and Diagrams
03 March	01:33:57	139:00:23	Rewriting theoretical part
03 March	01:58:12	140:58:35	Rewriting theoretical part
04 March	02:14:28	143:13:03	Rewriting theoretical part
05 March	01:44:05	144:57:08	Rewriting theoretical part
06 March	01:30:56	146:28:04	WKO-Präsentation vorbereiten
06 March	02:37:29	149:05:33	Rewriting theoretical part

figures/krems.pdf	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT		
	Krems		
	Abteilung:	Informationstechnologie	

07 March	04:36:11	153:41:44	Empirischer Teil Bilder erstellen
09 March	01:52:56	155:34:40	Dashboard Empire schreiben
09 March	00:34:49	156:09:29	Theoretisch Teil mit Prof. Katzenschlager besprechen
09 March	04:41:33	160:51:02	Rewriting theoretical part
13 March	05:57:24	166:48:26	Dashboard Empire schreiben
14 March	02:33:31	169:21:57	WKO-Präsentation vorbereiten
15 March	00:36:23	169:58:20	Empirischer Teil Angular schreiben
18 March	07:48:52	177:47:12	WKO Präsentation
18 March	00:42:20	178:29:32	Empirischer Teil Angular schreiben
20 March	00:55:25	179:24:57	Cytoscape schreiben
20 March	00:42:02	180:06:59	EWA
20 March	02:07:14	182:14:13	Dashboard Empire schreiben

Tabelle A.3.: Arbeitstagebuch Schneider

figures/krems	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
	Abteilung: Informationstechnologie

A.6. Datenträgerbeschreibung

Auf dem beigeleiten USB-Stick befinden sich folgende Dateien:

- Die schriftliche Arbeit befindet sich unter **thesis/documentation/final.pdf**
- Die schriftliche Arbeit in $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ -Format befindet sich unter **thesis/latex/**
- Sämtliche Dateien bezüglich des Projektmanagements befinden sich unter **assets/**
- Der gesamte Code befindet sich unter **prototype/**
- Eine Datei zum Ausführen der Applikation befindet sich in **prototype/docker-compose.yml**

Das ausschließliche Recht zur Nutzung, Verwaltung und Verfügung über den erstellten Programmcode liegt in Übereinstimmung mit den einschlägigen Vertragsvereinbarungen und geistigen Eigentumsrechten bei Siemens.

A.7. Ausführung des Prototypen

Die Ausführung unserer Applikation kann mithilfe der Dateien auf dem USB-Stick und der *Docker Engine* durchgeführt werden. Hierfür muss man einfach in das Verzeichnis **prototype/** wechseln und den Befehl des Quellcodes A.1 ausführen.

```
1 cd .\prototype\
2 docker compose up -d
```

Quellcode A.1: Applikation ausführen mittels Docker Compose

Damit anschließend auch Testdaten in der Datenbank verfügbar sind, muss ein Powershell-Script ausgeführt werden. Die Anweisung dafür ist im Quellcode A.2 zu finden.

```
1 cd .\prototype\test_data\
2 powershell.exe .\run_insert.ps1
```

Quellcode A.2: Testdaten in PostgreSQL einspielen