

DIPLOMARBEIT

Visualisierung der Ergebnisse des Stromnetzmodells

Ausgeführt im Schuljahr 2023/24 von:

Felix Schneider 5AHIT-19
Clemens Schlipfinger 5AHIT-17

Betreuer:

Ing. Jürgen Katzenschlager MSc, BEd
Ing. Jürgen Katzenschlager MSc, BEd

Krems, am 02. April 2024

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

DIPLOMARBEIT

DOKUMENTATION

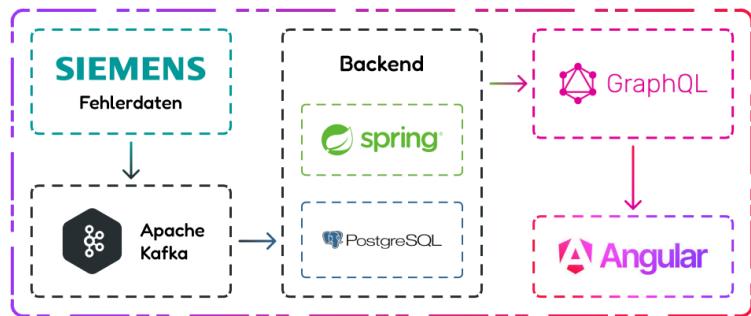
Namen der Verfasser/innen	Clemens Schlipfinger Felix Schneider
Jahrgang / Klasse Schuljahr	5AHIT 2023/24
Thema der Diplomarbeit	Visualisierung der Ergebnisse des Stromnetzmodells
Kooperationspartner	Siemens AG Austria

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	(Realisierung)
--------------	----------------

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

Typische Grafik, Foto etc.
(mit Erläuterung)



Diese Grafik zeigt den Architekturaufbau unseres Prototypen. Die Daten werden von der Siemens GNA über Kafka in die PostgreSQL Datenbank gespeichert. Von dort können Sie über eine GraphQL API vom Frontend abgerufen werden.

Teilnahme an
Wettbewerben,
Auszeichnungen

Bosch Innovationspreis 2024

Möglichkeiten der
Einsichtnahme in die Arbeit

...

Approbation
(Datum / Unterschrift)

Prüfer/in

Abteilungsvorstand /
Direktor/in

DIPLOMA THESIS

Documentation

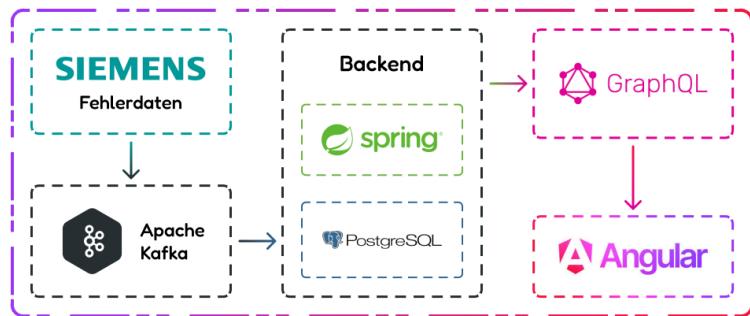
Authors	Clemens Schlipfinger Felix Schneider
Form	5AHIT
Academic year	2023/24
Topic	Application for a company
Co-operation partners	Siemens AG Austria

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	(Realisierung)
-------------	----------------

Results	The aim of the project is to develop a user-friendly web application with filter options, graphs and diagrams. A decentralised backend system is used, which can be smoothly integrated into the Siemens infrastructure and at the same time guarantees maximum stability. This makes it easier for Siemens engineers to analyse network model errors.
---------	--

Illustrative graph, photo (incl. explanation)



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.

Participation in competitions
Awards

Bosch Innovation Award 2024

Accessibility of diploma thesis

...

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.	Abstract	12
2.3.	Forschungsfragen	13
2.3.1.	Message Propagation	13
2.3.2.	Optimale Darstellungsmethoden	13
2.4.	Strukturierung der Arbeit	14
3.	Message Propagation	15
3.1.	Einleitung	15
3.1.1.	Service-Oriented Architecture	15
3.1.2.	Schlüsselfaktoren für verteilte Systeme	15
3.1.3.	Andere Kommunikationstechniken in verteilte Systeme	16
3.2.	Enterprise Messaging Systems	18
3.2.1.	Eigenschaften	18
3.2.2.	Enterprise Service Bus	19
3.2.3.	grundlegende Konzepte	19
3.2.4.	Komponenten eines Messaging Systems	20
3.2.5.	Messaging Models	21
3.2.6.	Consumption Mode	22
3.2.7.	Quality-of-Service Garantien	23
3.2.8.	Protokoll	23
3.2.9.	Latenz und Durchsatz	24
3.3.	Vergleich von populären Enterprise Messaging Services	25
3.3.1.	Apache Kafka	25
3.3.2.	RabbitMQ	27
4.	Graphentheorie	29
4.1.	Was ist ein Graph?	29
4.2.	Ausrichtung	29
4.3.	Konnektivität	29
4.4.	Zyklen	30
4.5.	Gewicht der Kanten	30
4.6.	Vollständigkeit	31
4.7.	Bipartition	31
4.8.	Planarität	32
4.9.	Eulerscher Graph	32
4.10.	Hamiltonscher Graph	33

	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
Abteilung:	Informationstechnologie

5.	Visualisierung	35
5.1.	Benutzerfreundlichkeit und Performance	35
5.1.1.	Intuitivität	35
5.1.2.	Interaktivität	36
5.1.3.	Performance	37
5.2.	Arten von Datendarstellungen	37
5.2.1.	Darstellung von einfachen Daten in Tabellen, Diagrammen und Kennzahlen	38
5.2.2.	Darstellung von komplexen Daten in Graphen, Heatmaps und Diagrammen	40
5.3.	Bibliothek zur visuellen Darstellung insbesondere von Graphen	45
6.	Architektur des Prototypen	46
6.1.	Kafka und das Backend	46
6.2.	GraphQL und das Frontend	46
6.2.1.	Die Schnittstelle zwischen Backend und Frontend: GraphQL	46
6.2.2.	Das Rahmenwerk aus der JavaScript Welt: Angular	46
6.2.3.	Die Bibliothek zum Erstellen des Graphen: Cytoscape	46
7.	Aufsetzen eines Kafka-Systems	49
7.1.	Herausforderungen einer GraphQL API mit relationalen Datenbanken	49
8.	Webapplikation	50
8.1.	Dashboard mit Kennzahlen des Netzmodells	51
8.2.	Tabellen mit Filter- und Suchfunktionen	52
8.3.	Graph des Stromnetzmodells	52
8.3.1.	Einfache Integration von Cytoscape in Angular	52
8.3.2.	Verwendung von Erweiterungen der Cytoscape-Bibliothek	54
9.	Bewertung	56
10.	Zusammenfassung und Ausblick	57
10.1.	Zusammenfassung	57
10.2.	Ausblick	57
I.	Literaturverzeichnis	58
II.	Abbildungsverzeichnis	63
III.	Tabellenverzeichnis	65
IV.	Quellcodeverzeichnis	66
V.	Abkürzungsverzeichnis	67
VI.	Übersetzungsverzeichnis	68
A.	Anhang	69
A.1.	Kapitelverzeichnis	69

A.2. Besprechungsprotokolle	70
A.2.1. Begleitprotokoll 1	70
A.2.2. Begleitprotokoll 2	73
A.2.3. Begleitprotokoll 3	76
A.2.4. Begleitprotokoll 4	79
A.2.5. Begleitprotokoll 5	82
A.3. Pflichtenheft	83
A.4. Arbeitspaket	93
A.5. Projekttagebücher	94
A.5.1. Projekttagebuch Clemens Schlipfinger	94
A.5.2. Projekttagebuch Felix Schneider	95
A.6. Datenträgerbeschreibung	97

htlkrems	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 MSc Jürgen Katzenschlager und Auftraggeber ist Siemens AG Austria.

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 Log Files gespeichert. Ein ausfallsicheres System mit strukturierter Visualisierung wird für einfache Auswertungen benötigt.

1.3. Abstract

htlkrems	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
Abteilung:	Informationstechnologie

1.4. Danksagung

Wir möchten uns insbesondere bei unserem Betreuer MSc Jürgen Katzenschlager bedanken. Dieser hat uns während der gesamten Umsetzung unserer Arbeit mit seiner Expertise unterstützt und uns wertvolle Ratschläge gegeben.

Des Weiteren möchten wir uns besonders bei unseren Familien und unseren Freunden bedanken, welche uns in vielerlei Hinsicht bei unserem Projekt unterstützt haben. Wir fanden bei ihnen immer ein offenes Ohr, welches uns emotional unterstützte, indem sie für uns da waren und uns zuhörten, wenn wir uns über unsere Herausforderungen und Sorgen im Zusammenhang mit unserem Projekt unterhalten haben. Ihre Unterstützung hat geholfen, die emotionalen Belastungen, die mit dem Projekt einhergingen, besser zu bewältigen.

1.5. Gendererklärung

Zur besseren Lesbarkeit der Diplomarbeit wurde ausschließlich die männliche Form verwendet. Da Begriffe wie "Benutzerinnen und Benutzer" den Text unleserlich machen, wurde es schlicht auf "Benutzer" gekürzt, dies soll jedoch keine Geschlechterdiskriminierung zum Ausdruck bringen.

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. Aufgrund der unzähligen Elementen, 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 gewissen Fehler, meistens in Form von Abweichungen von Sollwerten, im Netzmodell auftreten. Siemens verfügt aktuell über keine Applikation, welche dieser 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 ausfallsichere Applikation, welche diese stark vernetzten Daten mit optimalen Darstellungsarten visualisiert. Dabei gehen wir in dieser Arbeit besonders auf die Gestaltung eines entkoppelten Backend-Systems, welches im Prototypen mit Kafka umgesetzt wird, und einige geeignete Visualisierungsarten für solch komplexe Daten ein.

2.2. Abstract

Siemens is developing a system for many customers inside and outside Austria that constantly monitors and thus guarantees the reliability of the utility grid. Due to the countless elements, such as busbars, arresters, generators, transformers, disconnectors, circuit breakers, stations, fuses and load-break switches, this power network consists of extremely complex data. Furthermore, with so many elements, it can easily happen that certain errors, usually in the form of deviations from set values, occur in the network model. Siemens does not currently have an application that efficiently processes and visualises this error data. The engineers at Siemens analyse the errors using simple text files, which are difficult to read and unnecessarily increase the duration of the repair in the event of a major failure.

In order to make the analysis of fault data more efficient, Clemens Schlipfinger and Felix Schneider are writing a fail-safe application that visualises this highly networked data with optimal display types. In this work, we focus in particular on the design of a decoupled backend system, which is implemented in the prototype with Kafka, and some suitable visualisation types for such complex data.

htlkrems	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 muss die Visualisierung dieser Informationen spezifisch an die Anforderungen angepasst werden.

Daraus ergibt sich folgende Forschungsfrage:

Visualisierungsmethoden für stark vernetzte Daten

htlkrems	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
Abteilung:	Informationstechnologie

2.4. Strukturierung der Arbeit

Der Inhalt dieser Arbeit unterteilt sich in drei Hauptabschnitte. Jedes Kapitel ist eindeutig einem Abschnitt zuordenbar.

Um mit der Materie vertraut zu werden, werden bestehende Forschungen der Literatur zusammengetragen und aufgearbeitet. Anschließend wird die Konzeptionierung des Prototypen niedergeschrieben, welcher zum Beweisen der These herangezogen wird. Schlussendlich wird ein Katalog zur Bestimmung der richtigen Form von Message Propagation und Vor- und Nachteile der verschiedenen Visualisierungsmethoden bei stark vernetzten Daten erstellt.

3. Message Propagation

3.1. Einleitung

Die Anforderungen an Softwaresysteme wachsen exponentiell. Heutzutage umfassen sie eine Vielzahl an Aspekten, wie eine 24/7 Verfügbarkeit, Echt-Zeit Reaktionszeiten oder die Fähigkeit Millionen Benutzer gleichzeitig zu bedienen. Um diesen Voraussetzungen gerecht zu werden steigt die Komplexität der Anwendungssysteme. Die einst monolithischen Systeme müssen in kleinere Teile aufgebrochen werden und transformieren sich in verteilte Softwaresysteme (*Service-Oriented Architecture*). Diese kleineren Teile oder Dienste (*Services*) müssen untereinander kommunizieren und sich koordinieren, jedoch stellt sich diese Koordination als äußerst herausfordernd dar. Traditionelle Kommunikationstechniken wie *Remote Procedure Call (RPC)* können diesen Kriterien nicht gerecht werden und Anwendungen, welche diese Techniken verwenden werden zu schwer wartbare fehleranfällige Konstrukte. *Enterprise Messaging Systems* bieten eine geeignete Lösung für die Kommunikation in verteilten Softwaresystemen. [1]

3.1.1. 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 werden eine Sammlung von *Services* erstellt, die miteinander kommunizieren können, indem sie Service-Schnittstellen verwenden, um Nachrichten von einem *Service* an einen anderen zu übermitteln oder eine Aktivität zwischen einem oder mehreren *Services* zu koordinieren. Auf diese Weise kann mit SOA ein sehr flexibles System geschaffen werden. [2]

3.1.2. Schlüsselfaktoren für verteilte Systeme

Ein verteiltes System hat Faktoren, welche die Zuverlässigkeit, Effizienz und letztendlich die Qualität des Systems beeinflussen.

- **Kopplung:** In welchem Maße sind die Komponenten voneinander abhängig oder können unabhängig voneinander arbeiten?
- **Latenz:** Wie schnell erfolgt die Kommunikation zwischen den Systemen?
- **Skalierbarkeit:** Wie skaliert das gesamte System, wenn mehr Systeme hinzugefügt werden und die Arbeitslast eine Zunahme erfordert?
- **Wartbarkeit:** Wie aufwendig ist die Wartung der gesamten Systems?
- **Zuverlässigkeit:** Inwieweit kann das System konsistent und vertrauenswürdig seine Funktionen erfüllen und neigt es dazu, Fehler oder Ausfälle zu haben?
- **Verfügbarkeit:** Wie leicht kann das System ausfallen? Wie robust ist das System gegenüber Ausfällen von Komponenten?
- **Erweiterbarkeit:** Wie einfach ist es, neue Geschäftsszenarien (*Business Cases*) umzusetzen oder neue Komponenten zu integrieren?

3.1.3. Andere Kommunikationstechniken in verteilte Systeme

Es gibt eine Vielzahl an Techniken für die Kommunikation in verteilten Systemen, jedoch haben viele klare Nachteile gegenüber der Verwendung von *Messaging Systemen*.

3.1.3.1. Direkte Kommunikation

Die trivialste Lösung wäre eine Art von *Remove Procedure Calls* zwischen den einzelnen Teile des verteilten Systems, jedoch führt dies zu einer hohen Kopplung, da die einzelnen *Services* direkt miteinander kommunizieren. Wenn diese Systeme eine gewisse Größe erreichen werden sie zu undurchsichtige Netze an Abhängigkeiten, dies für zu einer Aufwendigen Wartung, niedrigen Skalierbarkeit und erschwere jede Art von Erweiterung des Systems. Jedoch können die Anforderungen an die Echtzeitkommunikation erfüllt werden. [1, 3]

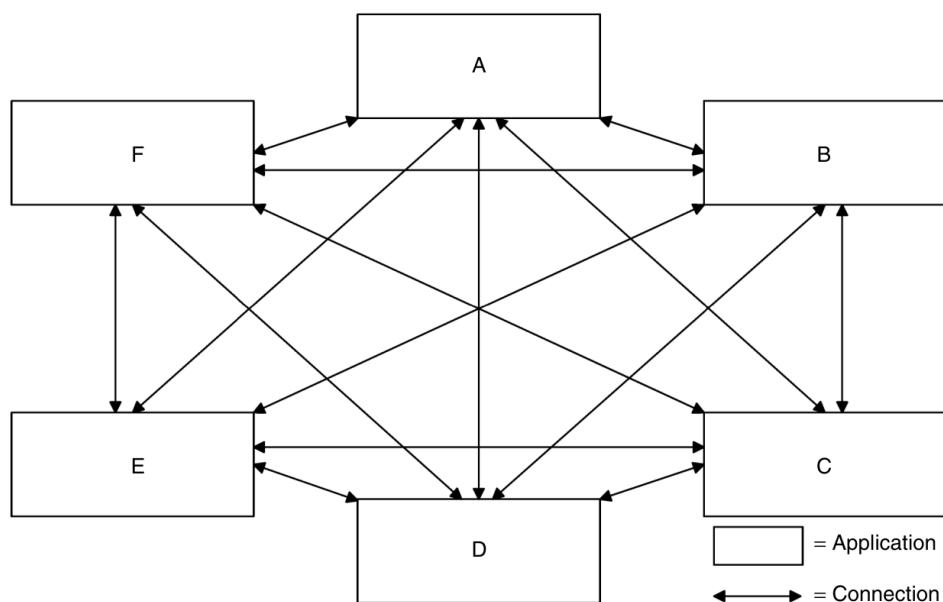


Abbildung 3.1.: Ein Beispiel für ein verteiltes System mit direkter Kommunikation
[1]

3.1.3.2. Verwendung einer 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. Weitere *Services* können auch leicht integriert werden, da sie nur einen Datenbank-Schema entsprechend kommunizieren müssen. Wenn jedoch das Datenbank-Schema angepasst wird, müssen alle *Services* angepasst werden, dieser Prozess könnte mit großen Aufwand verbunden sein. Eine hohe Skalierbarkeit zu erreichen ist in den meisten Fällen sehr herausfordernd, insbesondere bei relationalen Datenbanken, da diese von Natur aus schwer zu skalieren sind.[1, 3]

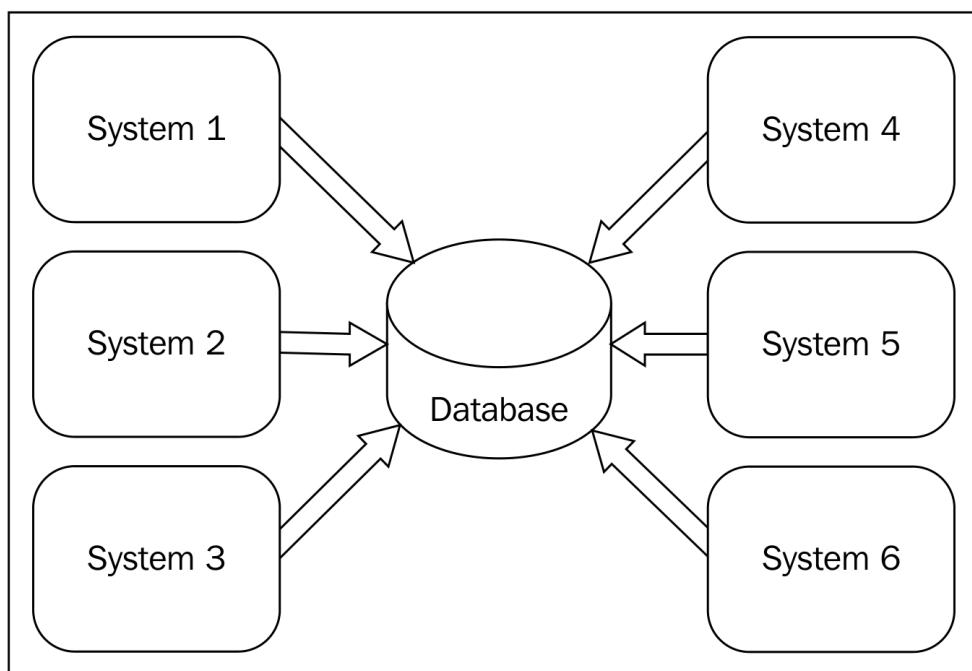


Abbildung 3.2.: System mit einer Datenbank als zentrale Kommunikationskomponente
[3]

3.2. Enterprise Messaging Systems

Durch ein *Messaging System* sind Sender (*Publisher*) und Empfänger (*Consumer*) stark entkoppelt, da sie über einen gemeinsamen Mittelmann - dem *Messaging System* - kommunizieren. Die Sender müssen Nachrichten dem *Messaging System* übertragen und die Empfänger erhalten die Nachrichten vom *Messaging System*. In vielen Fällen können die Kommunikationspartner nicht von einander wissen, dies reduziert die Kopplung auf das Minimum. Das *Messaging System* kann Nachrichten validieren, speichern, transformieren und zu den geeigneten Empfängern weiterleiten (*Routen*), dadurch wird ein einheitlichen Weg um die Kommunikation zwischen zwei Systemen zu ermöglichen geboten. [3]

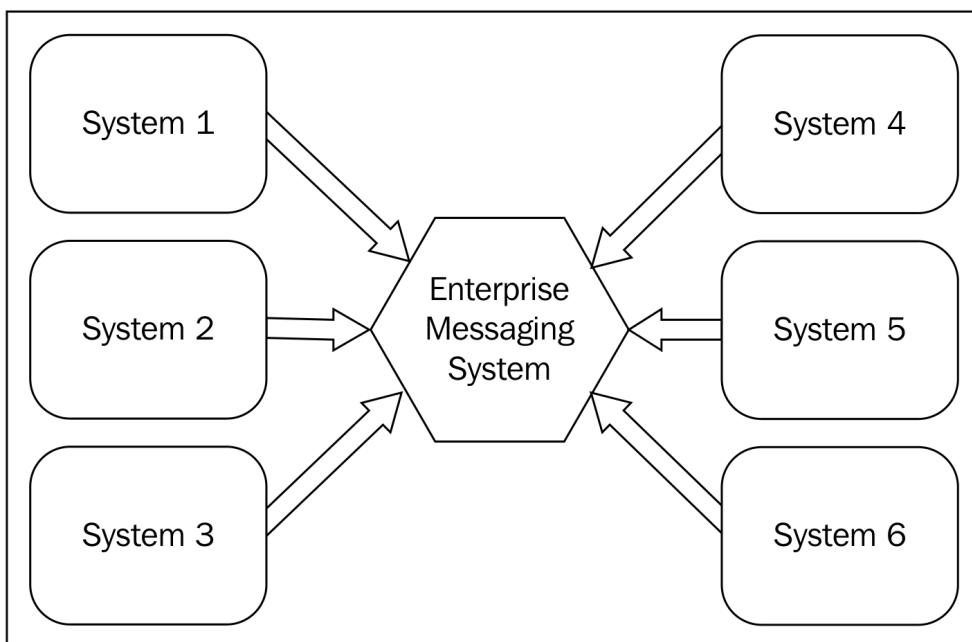


Abbildung 3.3.: Ein Beispiel für ein verteiltes System mit einem zentralen *Messaging System* [1]

3.2.1. Eigenschaften

Die folgenden Eigenschaften können bei verteilten Systemen, welche *Messaging Systeme* als Grundlage der Kommunikation verwenden beobachtet werden:

- **hohe Skalierbarkeit** - *Messaging Systeme* können sich an große Auslastungen anpassen durch *Clustering*
- **niedrige Koppelung** - Da das *Messaging System* einen Mittelmann darstellt, sind die Kommunikationspartner entkoppelt.
- **leichte Erweiterbarkeit** - Es besteht die Möglichkeit, dem *Messaging System* ohne Beeinträchtigung der bestehenden Kommunikationskanäle ein weiteres System hinzuzufügen.
- **niedrige Latenz** - *Messaging Systeme* sind auf Echt-Zeit Datenverarbeitung ausgelegt und verwenden viele Techniken um den niedrigsten *Overhead* wie möglich zu erreichen.

- **gute Wartbarkeit** - Das *Messaging System* bildet den zentralen Punkt der Kommunikation, was zu einer erheblichen Vereinfachung der Übersicht und der Fehlersuche führt.

3.2.2. Enterprise Service Bus

Wenn in einem System auch noch verschiedene Protokolle wie REST, SOAP, JMX, AMQP oder RPC verwendet werden und diese Protokolle auch in das *Messaging System* integriert werden. Die Integration könnte durch verschiedene Adapter realisiert werden, welche die Protokolle auf andere konvertieren. Wenn ein *Message System* diese weiteren Funktionalitäten besitzt und einer seiner Hauptaufgaben die Schaffung einer Kompatibilitätsschicht wird, dann wird von einem *Enterprise Service Bus (ESB)* gesprochen. Dadurch kann der ESB die Integration von bestehenden Systemen oder Systemen die an ein bestimmtes Protokoll gebunden sind deutlich erleichtern. Allerdings erfordert es erheblichen Aufwand, dass der ESB stets allen Kommunikationsanforderungen (Protokolle usw.) gerecht wird. [3, 4]

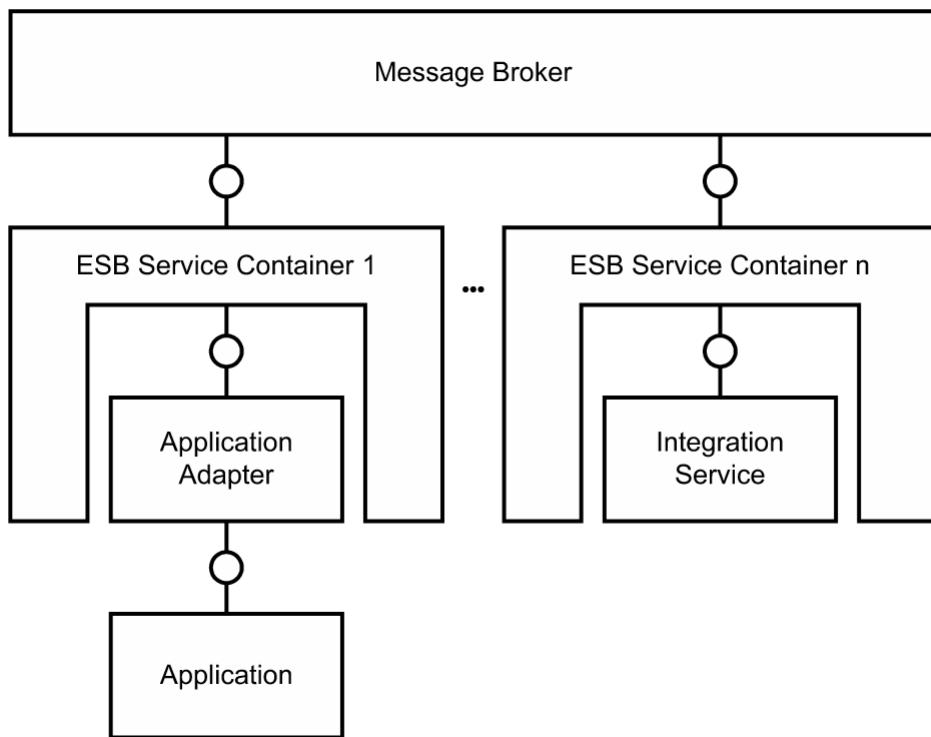


Abbildung 3.4.: Ein einfacher *Enterprise Service Bus* [4]

3.2.3. grundlegende Konzepte

3.2.3.1. Das zentrale Element - Nachrichten

Die Nachricht (*Message*) ist das zentrale Element der Kommunikation im *Messaging System*. Eine Nachricht hat typischerweise folgende Bestandteile:

- *Header* - enthält Metadaten, wie das *encoding*, *routing information* und sicherheitsrelevante Daten
- *Body* - enthält die eigentlichen Daten

[3]

3.2.3.2. Queues

Die *Message Queue* ist ein essenzieller Teil des *Messaging Systems*. Sie bieten die Funktionalität Nachrichten im *Messaging System* zu speichern. Die Sender reihen Nachrichten in die *Queue* ein und die Empfänger nehmen die Nachrichten über die *Queue* entgegen. Die standardmäßigen *Queues* folgen den *First-In First-Out (FIFO)* Prinzip. Die Sender und Empfänger können dabei völlig unabhängig voneinander mit der *Queue* interagieren.

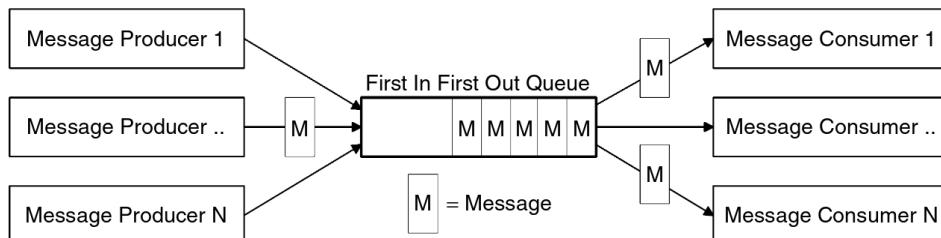


Abbildung 3.5.: Eine *Queue* [1]

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* ist die *Dead-Message Queue*, sie beinhalten Nachrichten, welche abgelaufen oder nicht Zustellbar (z.B. nicht gültiger Name für die Ziel-Queue) sind. [1]

3.2.4. Komponenten eines Messaging Systems

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

- Erhalt und Zustellen der Nachrichten
- Speichern der Nachrichten
- *Routing* von Nachrichten

Der *Cluster* ermöglicht den *Messaging System* ausfallsicher und skalierbar zu sein, da er aus einem Verbund von mehreren *Broker* besteht. Im *Cluster* werden die Nachrichten repliziert und verteilt, dafür gibt es zwei Architekturen:

- **Master-Slave** - Die *Broker* sind in *Master* und *Slave* unterteilt. Der *Master* stellt den Dienst zur Verfügung und die *Slaves* dienen nur als Backup.
- **Peer-to-Peer** - Die *Broker* haben alle den selben Status, jede Nachricht ist in mehreren *Brokers* abgespeichert.

Manche *Messaging Systeme* unterstützen auch das Zusammenspiel von mehreren *Clustern*, dass hat folgende Vorteile:

- Segregation der Daten
- Isolierung gemäß Sicherheitsanforderungen
- Mehrere Rechenzentren - Notfallwiederherstellung (*disaster recovery*)

[5, 6]

3.2.5. Messaging Models

Ein *Messaging System* verwendet eines oder mehrere Kommunikationsmodelle, welche die Grundlage für die Kommunikation der zu verbindenden Systeme definiert. [1]

3.2.5.1. Point-to-Point

In einer *Point-to-Point* Kommunikationsmodell gibt es nur einen Sender und einen Empfänger. Im Fall das es mehrere Empfänger für eine bestimmte Nachricht gibt, kann nur ein einziger sie erhalten, solche Empfänger werden auch *Competing Consumers* genannt. Diese Technik kann für eine einfache Art von *Load Balancing* verwendet werden, denn es werden die Nachrichten auf mehrere Empfänger aufgeteilt. [1]

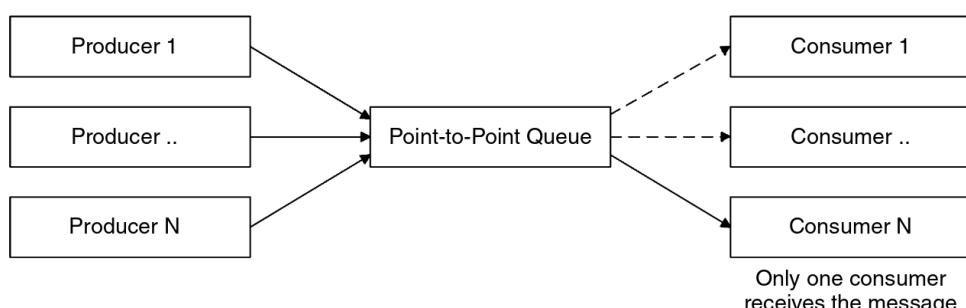


Abbildung 3.6.: Das *Point-to-Point Messaging Model* [1]

3.2.5.2. Publish-Subscribe

In der Form von *Publish-Subscribe* gibt es mehrere Sender und Empfänger. Die Nachrichten werden immer zu allen Empfängern geschickt. Das *Messaging System* leitet die Nachrichten an *Clients* weiter, basierend auf den *Topics*, für die sie sich angemeldet haben und an denen sie interessiert sind. Empfänger können sich für den Erhalt von den Nachrichten eines bestimmten *Topic* melden (*subscribe*) und Sender können Nachrichten an ein *Topic* senden (*publish*).

Jedoch gibt es keine Einschränkung für die Rolle des *Clients*; ein Client kann ein Empfänger und Sender eines *Topic* sein. [1]

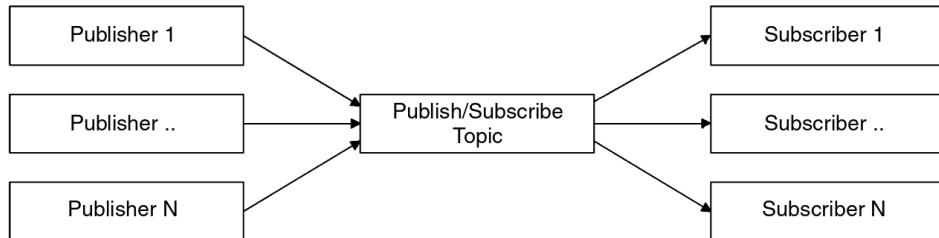


Abbildung 3.7.: Das *Publish-Subscribe Messaging Model* [1]

3.2.5.3. Request-Response

In einem Request-Response Kommunikationsmodell gibt es einen Sender und einen Empfänger. Wenn der Sender eine Nachricht schickt muss er auf die Antwort des Empfängers warten, welches zu einer höheren Kopplung und weitern Nachteilen führt. Durch diese Nachteile findet das Request-Response Kommunikationsmodell nur in speziellen Fällen Anwendung in *Messaging Systeme*. [1]

3.2.6. Consumption Mode

Dieser Aspekt der Kommunikation trifft nur auf die Empfänger zu und beschreibt, wie die Empfänger die Daten erhalten.

3.2.6.1. Push

In diesem Modus verteilt das *Messaging System* selber die Nachrichten an die Empfänger. Dieser Modus hat bessere Echtzeit-Performance, aber benötigt ein Flow-Control Mechanismus. Ein Problem bei diesen Modus ist, dass das *Messaging System* den Empfänger überlasten könnte, wenn in kurzer Zeit sehr viele Nachrichten überträgt. [1, 6]

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 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. [1, 6]

3.2.7. Quality-of-Service Garantien

Als Middleware System muss das *Messaging System* bestimmte Garantien zusichern, damit eine zuverlässige Operation eines verteilten Systems ermöglicht wird. [6]

3.2.7.1. Zustellungsgarantien

Damit die verlässliche Übertragung zwischen *Consumer* und *Producer* gewährleistet ist.

- **At-Most-Once** - Eine *Message* niemals wiederholt übertragen werden, jedoch bei dem Übertragungsvorgang verloren gehen.
- **At-Least-Once**: Eine Nachricht wird niemals nicht vollständig übertragen werden, kann aber wiederholt übertragen werden.
- **Exactly-Once**: Jede Nachricht wird exakt einmal übertragen.

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. [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* sehr ressourcenintensiv.

- **No-Ordering**: Es gibt keine Ordnung. Das ist ideal für eine hohe Performance.
- **Partition-Ordering**: Es gibt eine Ordnung für die Nachrichten in einem *Broker*, aber nicht über mehrere *Broker* hinweg.
- **Global-Ordering**: Die Nachrichten sind über den ganzen *Cluster* geordnet, jedoch hat dies eine große negative Auswirkung auf die Performance.

[6]

3.2.8. Protokoll

Das verwendete Protokoll, welches die Grundlage der Informationsaustausches bildet, 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 einen bestimmten *Messaging System* aufheben. *Closed-source* protocols sind oft Produkte von Veränderungen an existierenden *open-source* Protokollen oder völlig neue geschaffene Kommunikationsdefinitionen, welche nur für bestimmte Systeme gedacht sind. [6]

3.2.9. Latenz und Durchsatz

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

- Die Dauer für die *Metadaten-Verarbeitung* eines Packets, wie z.B. die Validierung oder das *routing*.
- Die Zeit, welche in Anspruch genommen wird für die *Replikation* eines Packets. 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 der Zugriff auf *RAM*-gespeicherte Daten sind weniger Zeit intensiv als ein Datenzugriff auf eine Festplatte.
- Weitere Verzögerungen werden durch die Einhaltung der *Quality-of-Service* Garantien erzwungen, wie beispielsweise die Ordungsgarantien, welche einen besonderen Effekt haben.

Low Latency Messaging Services minimieren diese Aspekte und können dadurch eine annähernd Echt-Zeit Übertragung ermöglichen. Der Durchsatz steht für die Menge an Daten, welche pro Zeiteinheit über das *Messaging System* übermittelt werden können. Damit die höchsten Durchsatzraten entstehen 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, denn das 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 [5] je nach Anwendungsfall bestimmen wie viele Nachrichten akkumuliert werden. [6]

3.3. Vergleich von populären Enterprise Messaging Services

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 platform* bezeichnet, da es den benötigten Durchsatzrate erreichen kann und eine Reihe an *stream processing* Funktionalitäten besitzt, wie zum Beispiel *Kafka Streams* oder *KSQL*. *Stream processing* ist die Datenverarbeitung an einen Fluss von Daten, also eine unendliche Serie an Daten. Außerdem verfügt Apache Kafka über *Connectors*, die zur Einbindung von externen Datenquellen oder Datensenken 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
- hohe Verfügbarkeit
- hohe Durchsatzraten
- Aufbewahrung der Nachrichten auf der Festplatte

Die *disk-based Retention*, also die Speicherung der Nachrichten auf der Festplatte bringt den großen Vorteil mit sich, dass Nachrichten über einen längeren Zeitraum im *Messaging System* aufbewahrt werden können. Da die *Messages* auf der Festplatte persistiert werden und nicht nur im RAM, kann kein Datenverlust auftreten. Es kann zum Beispiel während Wartungsarbeiten an den Empfängern trotzdem sicher gestellt werden, dass keine Nachrichten verloren gehen. [5]

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 *Topic* wird in mehreren Partition aufgeteilt. Diese Partitionen werden auf mehrere *Message Broker* verteilt und repliziert, da Apache Kafka normalerweise als *Cluster* arbeitet. Partitionen werden verwendet um die Nachrichten gleichmäßig auf die *Message Broker* aufzuteilen und den Empfängern gleichzeitiges Auslesen eines *Topics* zu ermöglichen. Wie es in der Abbildung 3.9 zu sehen ist, verwendet Kafka die *Peer-to-Peer-Replikationsmethode* und Empfänger erhalten Nachrichten über den *Consumption Mode Pull*. 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 3.8), aber die *Messages* werden auf die Empfänger aufgeteilt, somit bieten sich *consumer groups* perfekt für *Load Balancing* Anwendungen an. *Zookeeper* wird zum Koordinieren der *Consumer* und *Broker* verwendet, damit man in Kafka internes *Load Balancing* erreicht.

Apache Kafka unterstützt die Zustellungsgarantie *at-least-once* standardmäßig und die *at-most-once* Garantie kann beim *Producer* eingestellt werden. Dass die *exactly-once* Zusicherung eingehalten werden kann, müssen aber Veränderungen am Ziel-Storage System bewerkstelligt

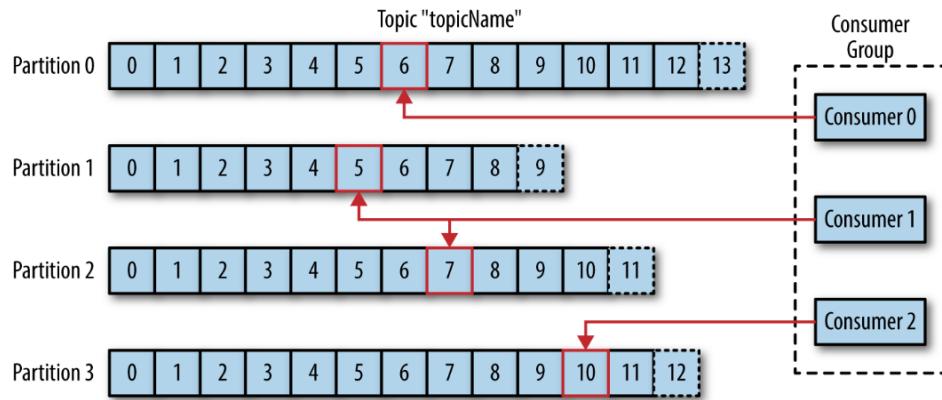


Abbildung 3.8.: Eine *consumer group* verarbeitet gemeinsam ein Topic. [5]

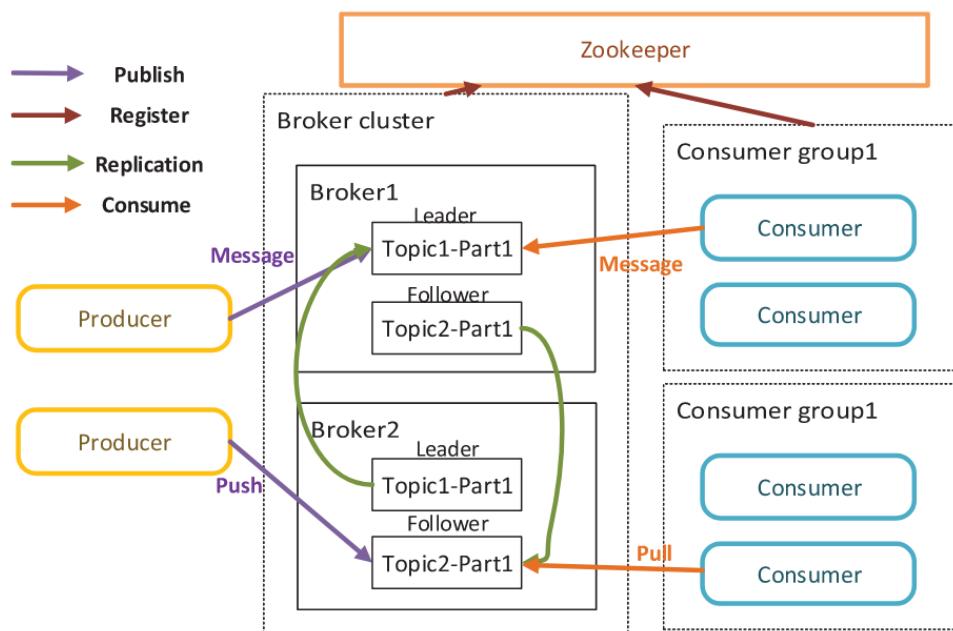


Abbildung 3.9.: Die Architektur des Apache Kafka Systems [6]

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

- Die Nutzung von *Zero-Copy*-Methoden um überflüssiges Kopieren von Daten zu vermeiden.
- Kafka überträgt Nachrichten in *Batches*. (*batch processing*)
- Nachrichten werden komprimiert um eine effiziente Datenübertragung zu ermöglichen.

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 3.10 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*, ist für hohe Auslastungen nicht gut geeignet. *Exchanges* sind für das *Routen* der Nachrichten benötigt, sie können komplexe *Routing*-Anforderungen umsetzen, wie zum Beispiel die Verteilung der Nachrichten auf mehrere *Queues*. Das RabbitMQ System ist ein *general-purpose Message Broker*, da es sehr viele Funktionalitäten implementiert, darunter befinden sich:

- *Push / Pull Consumption Mode*
- verschiedene standardisierte Protokolle wie *AMQP*
- *Point-to-Point* und *Publish-Subscribe Messaging Models*
- *Disk Nodes* oder *Memory Nodes* für *RabbitMQ Cluster*
- *At-least-once* oder *at-most-once* Zustellungsgarantien

RabbitMQ glänzt in den vielseitigen Funktionalitäten, die es anbietet, jedoch kann es mit der hohen Skalierbarkeit und Performance, insbesondere dem Datendurchsatz von Apache Kafka nicht mithalten. [6, 3]

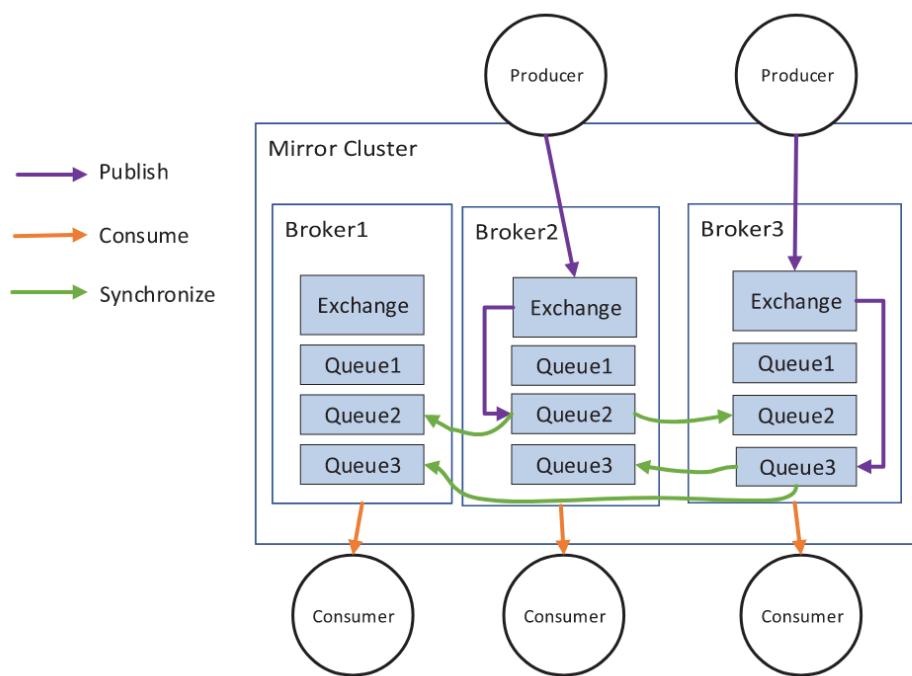


Abbildung 3.10.: Die Architektur des RabbitMQ Systems [6]

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. Was ist ein Graph?

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]

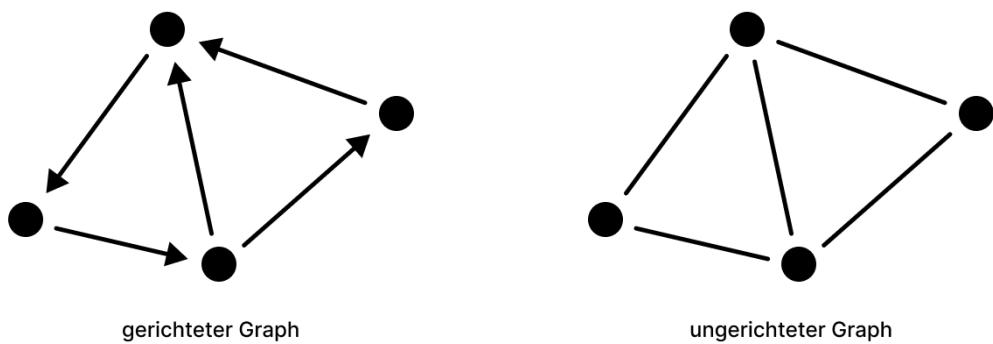


Abbildung 4.1.: Zwei Graphen dessen Unterschied die Ausrichtung ist

4.3. Konnektivität

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

Teilgraphen 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]

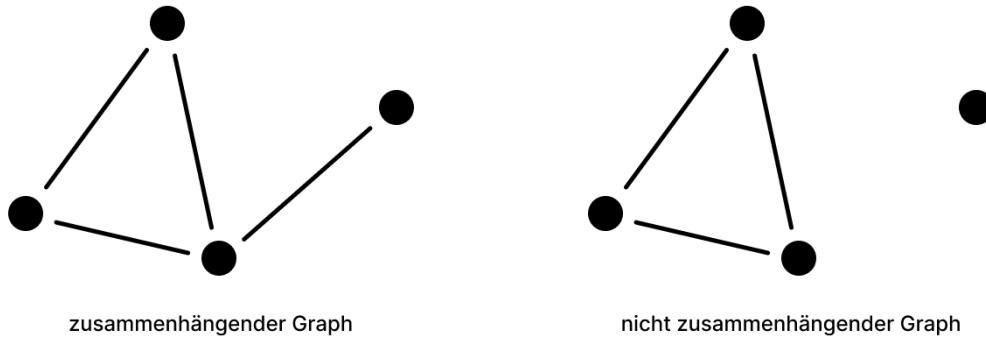


Abbildung 4.2.: Ein Graph all dessen Knoten verbunden sind (links); ein Graph, welcher nicht verbundene Knoten enthält (rechts)

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]

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 Verbindungsstrecke zweier Haltestellen bei einem öffentlichen Verkehrsnetzwerk. [8]

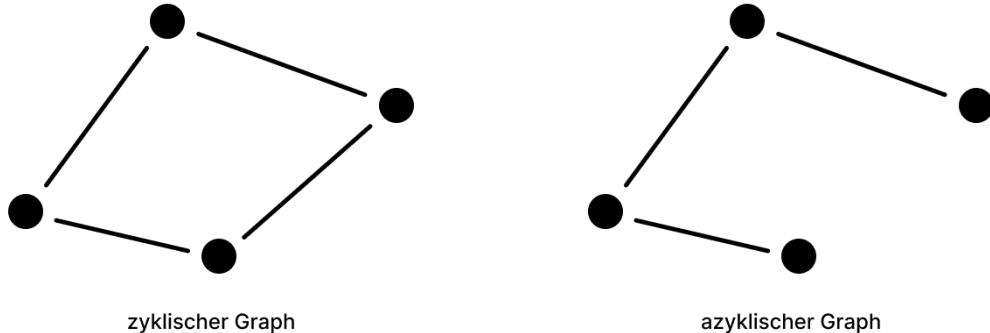


Abbildung 4.3.: Ein Graph, welcher keinen geschlossenen Kreis (Zyklus) enthält (links); ein Graph mit Zyklus (rechts)

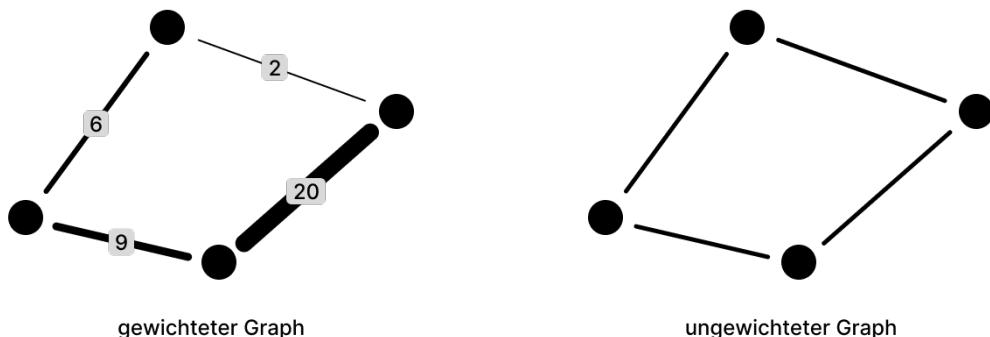


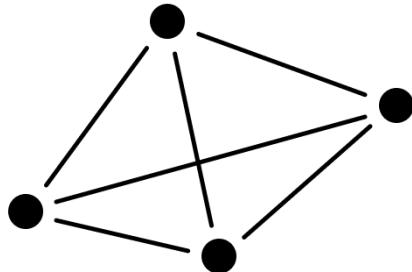
Abbildung 4.4.: Ein Graph, welcher gewichtete Kanten hat (links); ein Graph, dessen Kanten immer das gleiche Gewicht haben (rechts)

4.6. Vollständigkeit

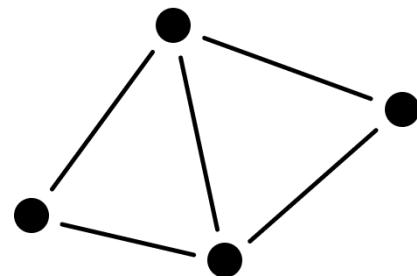
Ein vollständiger Graph ist durch die Eigenschaft gekennzeichnet, dass jeder Knoten mit jedem anderen Knoten durch eine Kante verbunden ist. Diese charakteristische Vollständigkeit der Verbindungen impliziert eine maximale Interaktion zwischen den einzelnen Knoten des Graphen, wodurch sämtliche möglichen Kantenrelationen realisiert sind. Diese Einschränkung der Flexibilität von den Eigenschaften eines Graphen kommen aus diesem Grund seltener vor, weswegen die meisten Graphen unvollständig sind. [8]

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]

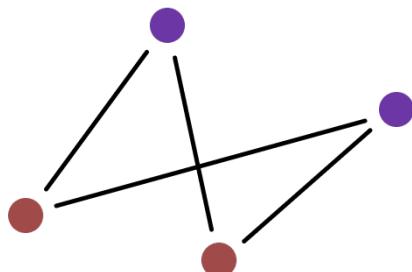


vollständiger Graph

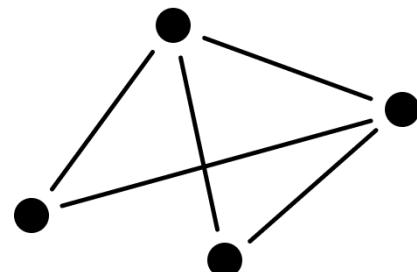


unvollständiger Graph

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



bipartiter Graph



nicht bipartiter Graph

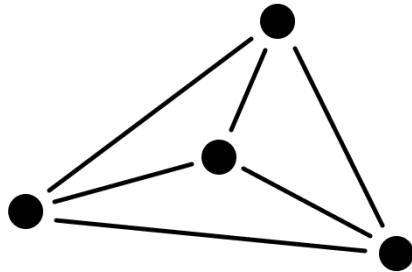
Abbildung 4.6.: Graph, welcher zwei Mengen ohne inhärente Verbindungen hat (links); keine Bipartition möglich beim rechten Graph

4.8. Planarität

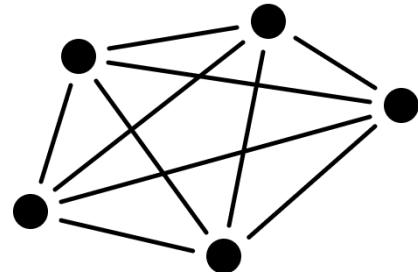
Ein planarer Graph kann auf einer Ebene ohne Kantenkreuzungen dargestellt werden, was eine klare zweidimensionale Visualisierung ermöglicht. Im Gegensatz dazu erfordert ein nicht planarer Graph Kantenkreuzungen bei der Darstellung auf einer Ebene, was die visuelle Erfassung erschwert. Die Planarität beeinflusst somit nicht nur die Struktur, sondern auch die graphische Repräsentation und Analyse des Graphen [10, 11].

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].

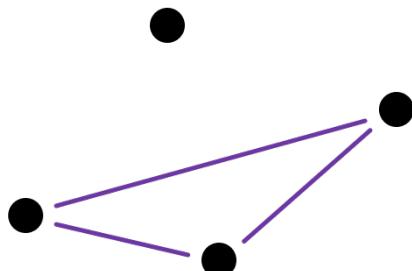


planarer Graph

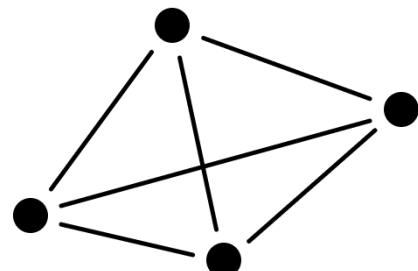


nicht planarer Graph

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



eulerscher Graph



nicht eulerscher Graph

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]

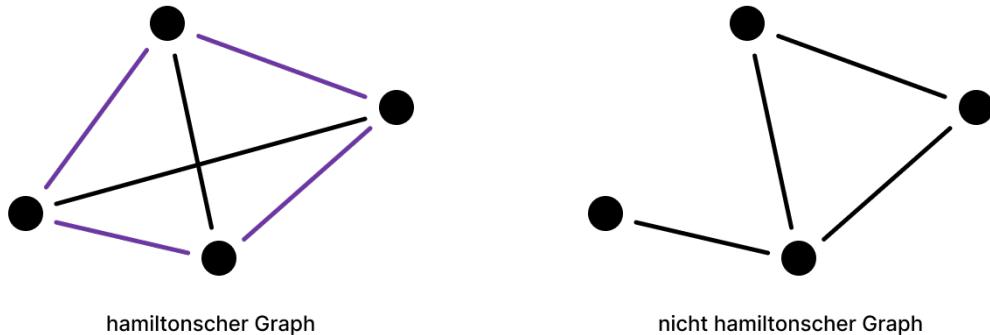


Abbildung 4.9.: Graph mit Zyklus, welcher alle Knoten beinhaltet (links); kein Hamiltonscher Kreis möglich (rechts)

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 Checkboxen, 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 den 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]

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. [22]

«Interaktivität ist das Potenzial eines technischen Einzelmediums oder einer Kommunikationssituation, das interaktive Kommunikation begünstigt, also den Prozess der Interaktion.» - *Christoph Neuberger* [23]

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. [24]

Eine wichtige Komponente ist die Anpassungsfähigkeit der Filter, um unterschiedlichen Anforderungen gerecht zu werden. Die Möglichkeit, Filterkriterien zu kombinieren oder anpassbare 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. [25]

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. [26, 24]

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. [27]

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. [28, 29, 30, 31]

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 Nutzererfahrung dienen. [29, 32]

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. [33]

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. [34]

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

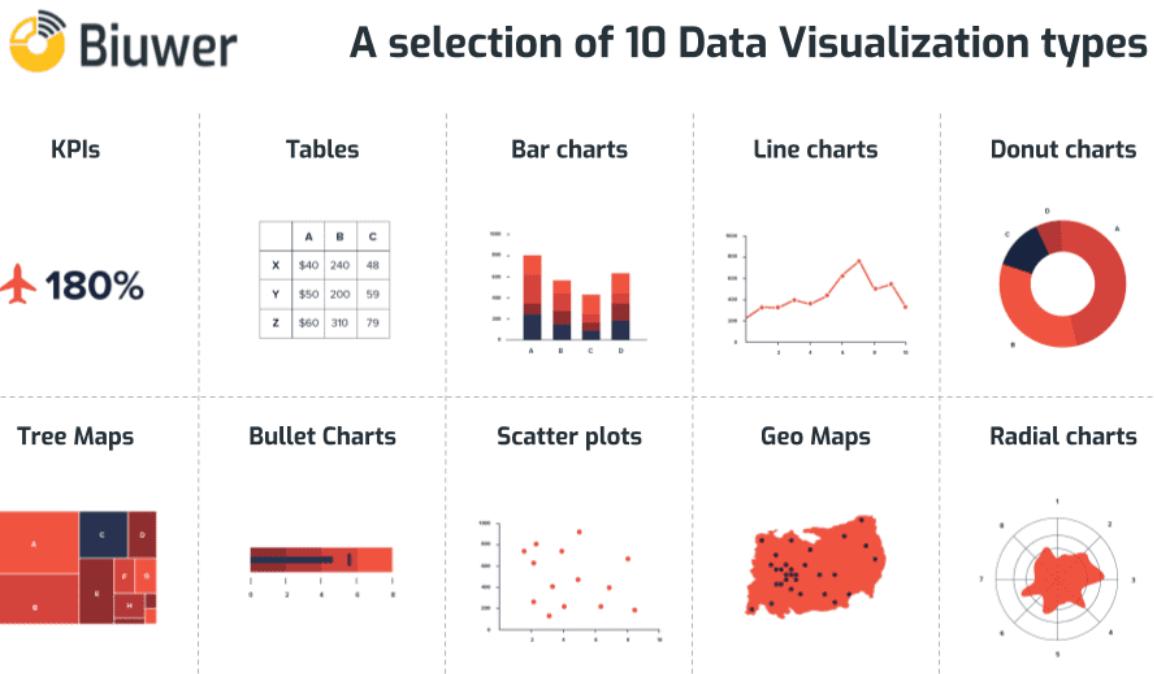


Abbildung 5.1.: Einfache Datenvisualisierungen in Form von Werten, Diagrammen, Tabellen und Karten [35]

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 Visualisierung gewarnt, welche vom wesentlichen Grundgedanken der Darstellung ablenken würden. [36]

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. [36]

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. [37]

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. [38]

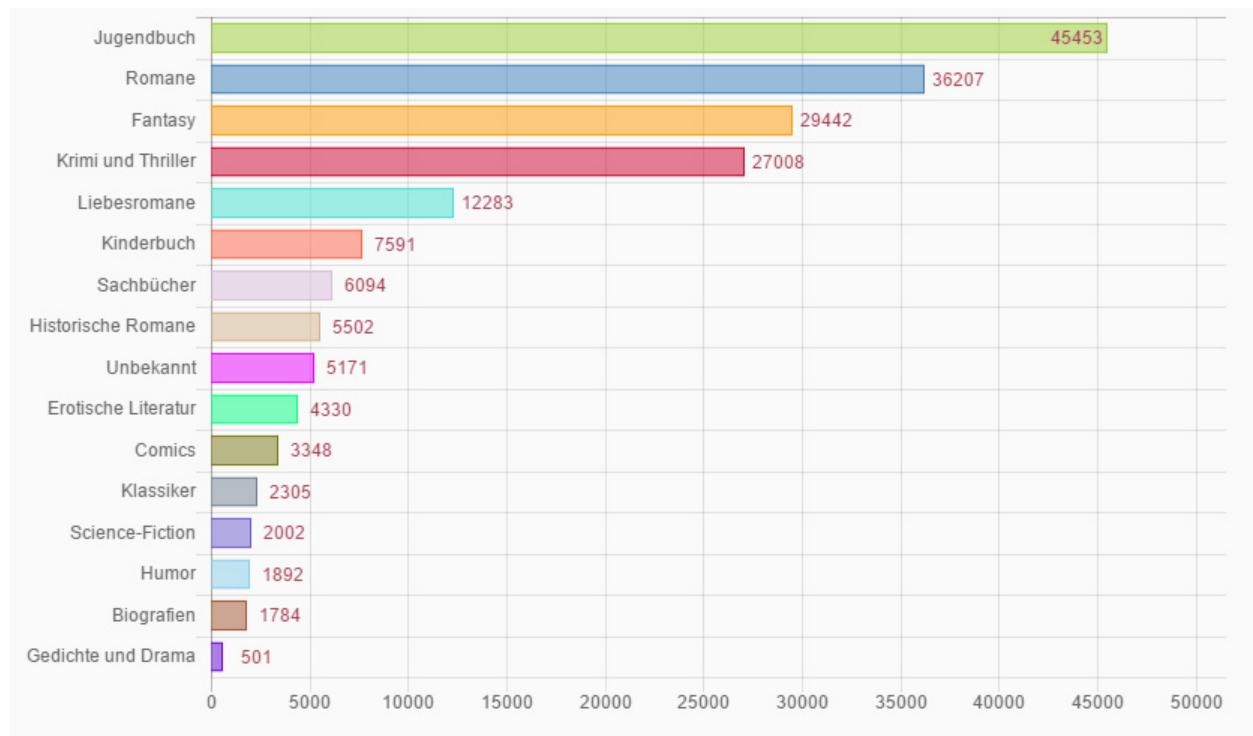


Abbildung 5.2.: Beliebtheit der verschiedenen Buchgenre im Jahr 2016 [39]

Jede Diagrammart hat demnach ihre Daseinsberechtigung und soll auch für den spezifischen Anwendungsfall herangezogen werden. [38]

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*. [34, 40]

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. [40, 41]

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 Darstellung 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 Anbetacht dieser Grafik vergessen werden.

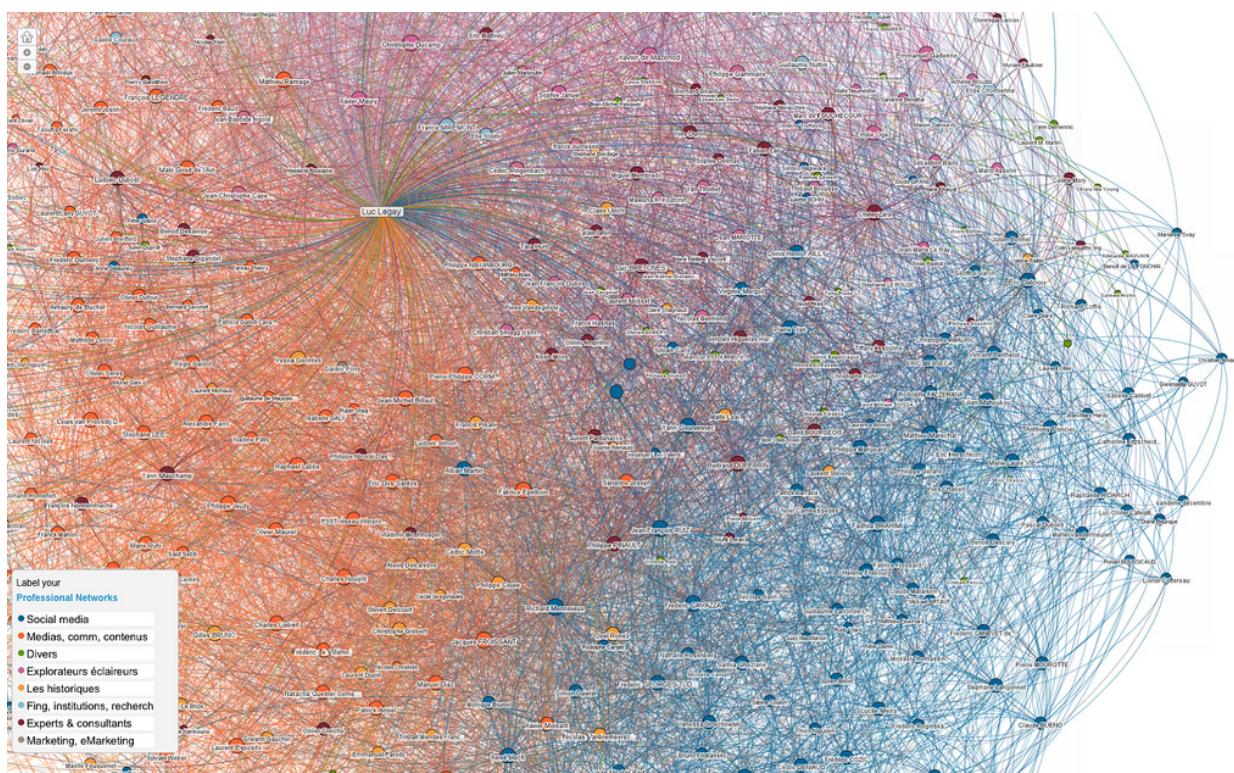


Abbildung 5.3.: Stark vernetzte Daten in einem Graphen (LinkedIn Netzwerk) [42]

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. [41]

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. [43]

Die Algorithmen hinter *Graph Clustering* sind unterstützt von maschinellem Lernen. Die künstliche Intelligenz versucht hierbei Gemeinsamkeiten der einzelnen *Nodes* innerhalb des Graphen zu finden. Diese Ähnlichkeiten können entweder direkte Eigenschaften der Knoten 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. [43]

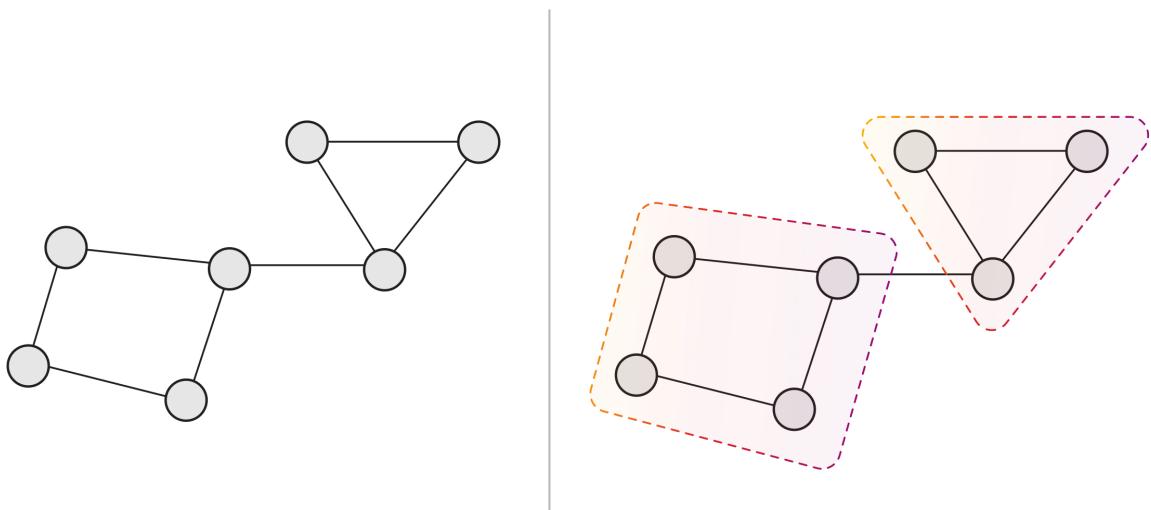


Abbildung 5.4.: *Clustering* eines Graphen [43]

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 Jahren rundherum aggregiert dargestellt sind. [41]

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. [44, 45, 46]

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.

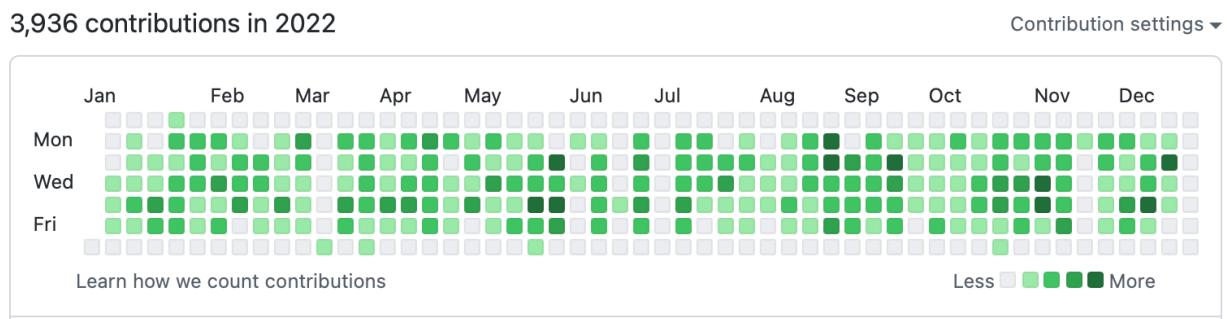


Abbildung 5.5.: *Heatmap* der Beiträge eines GitHub Benutzers im Jahr 2022 [44]

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. [47]

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

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.

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.

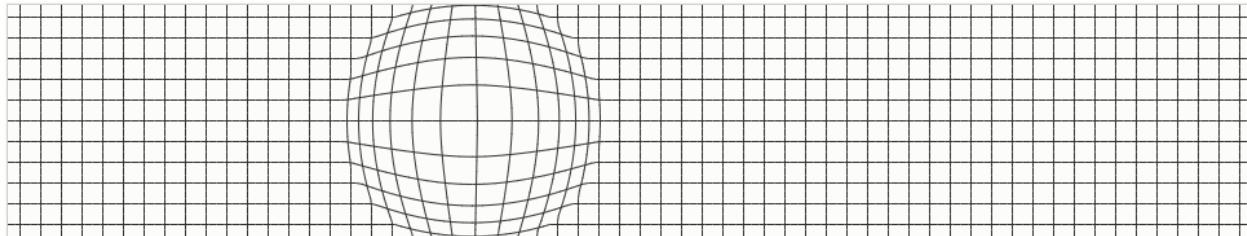


Abbildung 5.6.: Demonstration der Lupentechnik „Fisheye“ auf einem Grid [47]

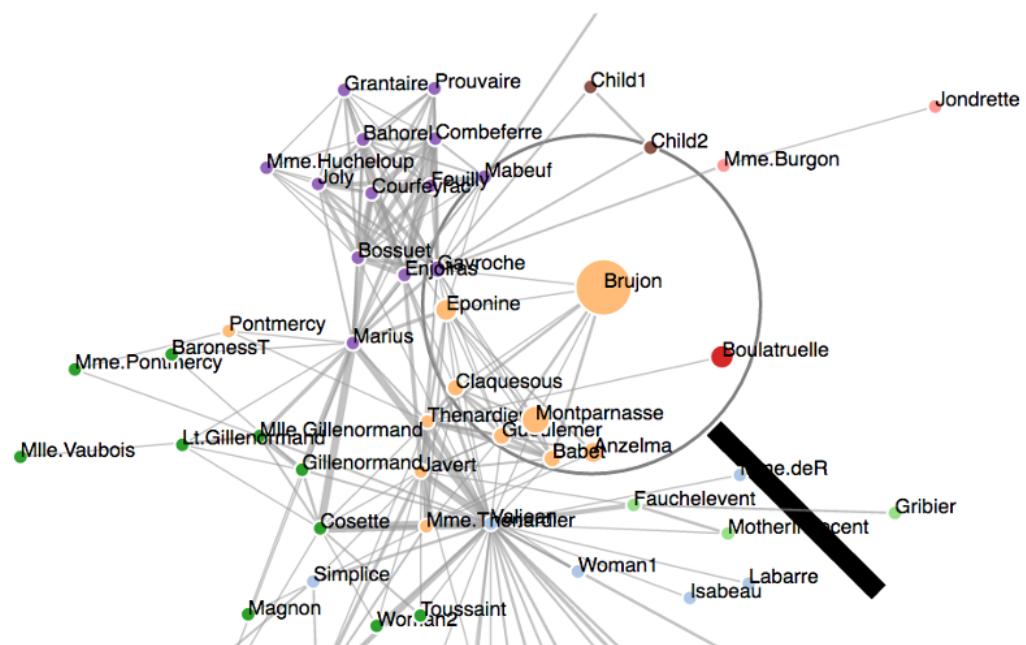


Abbildung 5.7.: Demonstration des Fischauges in einem Graphen [Source: GitHub Gist]

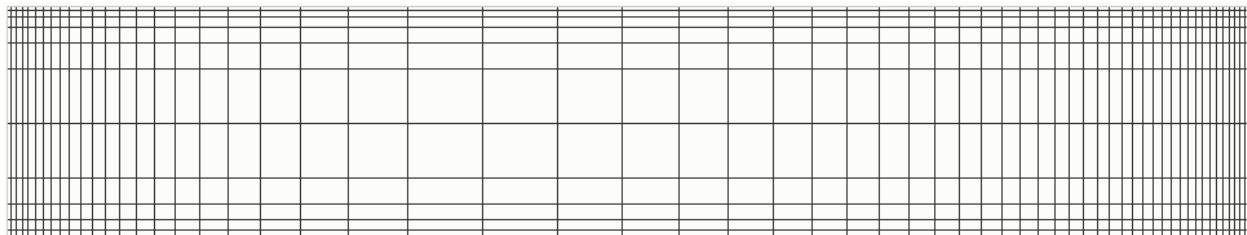


Abbildung 5.8.: Demonstration der Lupentechnik „Kartesisch“ auf einem Grid [47]

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. [41, 48]

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 *Heatmap*¹ herangezogen, welche in Abbildung 5.9 zu Demonstrationszwecken abgebildet ist.



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

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

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 biomolekularen 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.

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* 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.

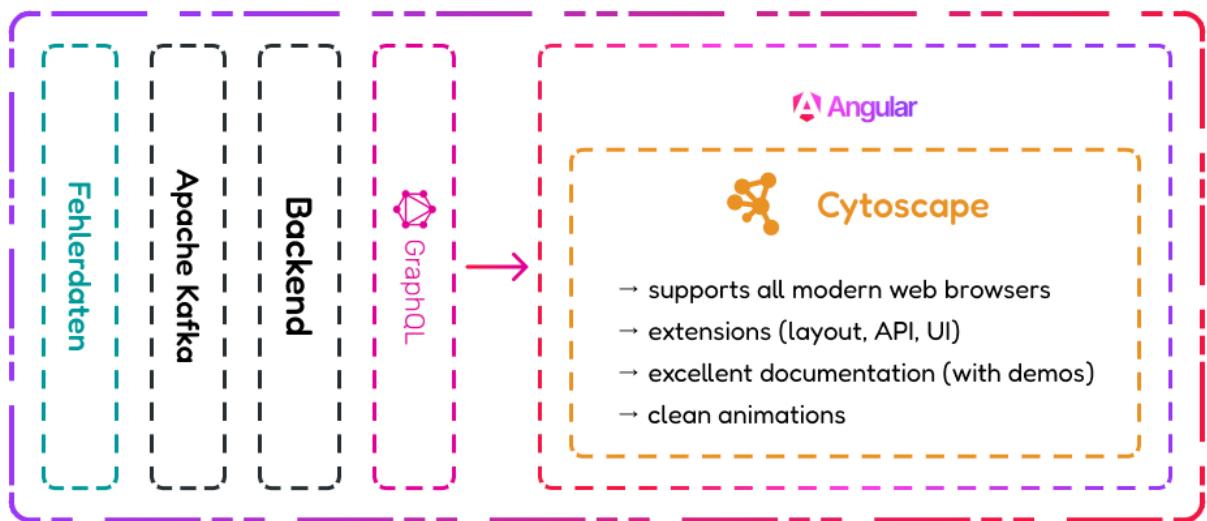


Abbildung 5.10.: Architektur unseres Prototypen mit Fokus auf die Bibliothek *Cytoscape*

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 in den ausfallsicheren *Enterprise Service Bus 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.

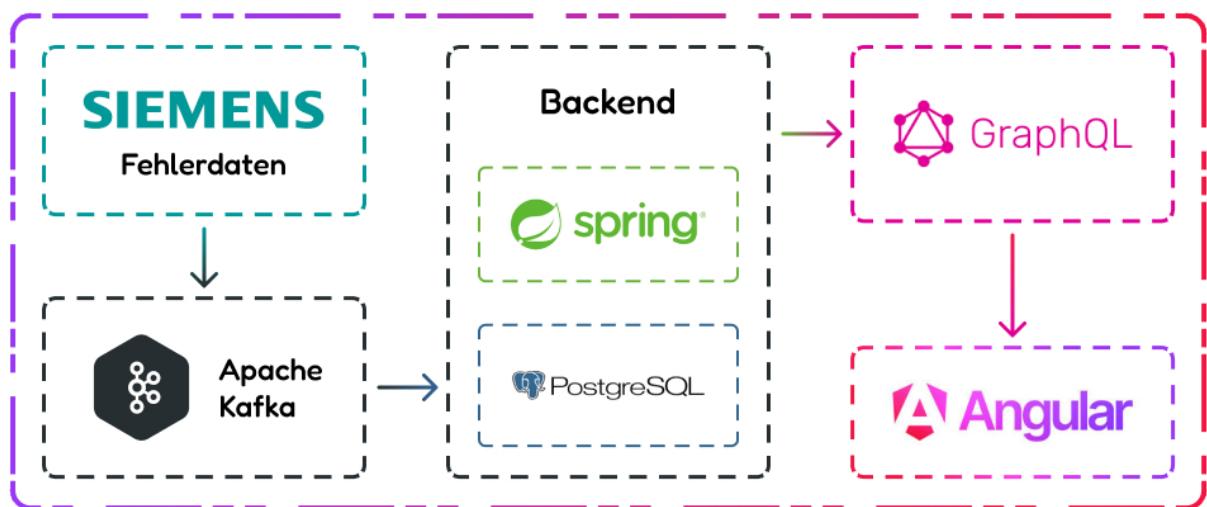


Abbildung 6.1.: Übersicht der Architektur unseres Prototypen

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

6.1. Kafka und das Backend

6.2. GraphQL und das Frontend

Diese Sektion erklärt kurz die technische Ausimplementierung 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 *Library Cytoscape* verwendet.

6.2.1. Die Schnittstelle zwischen Backend und Frontend: GraphQL

6.2.2. Das Rahmenwerk aus der JavaScript Welt: Angular

6.2.3. Die Bibliothek zum Erstellen des Graphen: Cytoscape

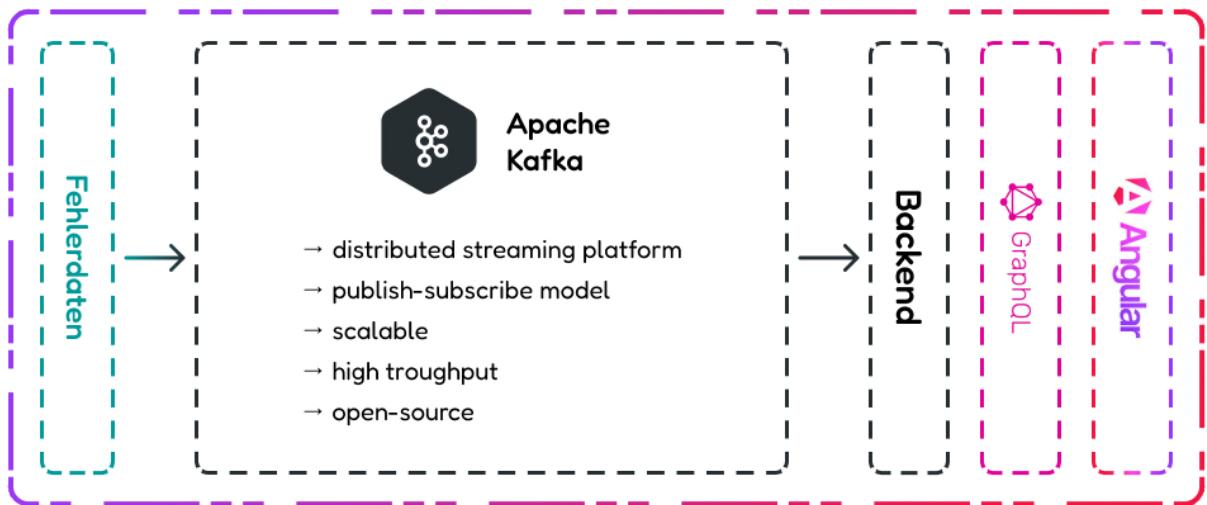


Abbildung 6.2.: Architektur unseres Prototypen mit Fokus auf Kafka

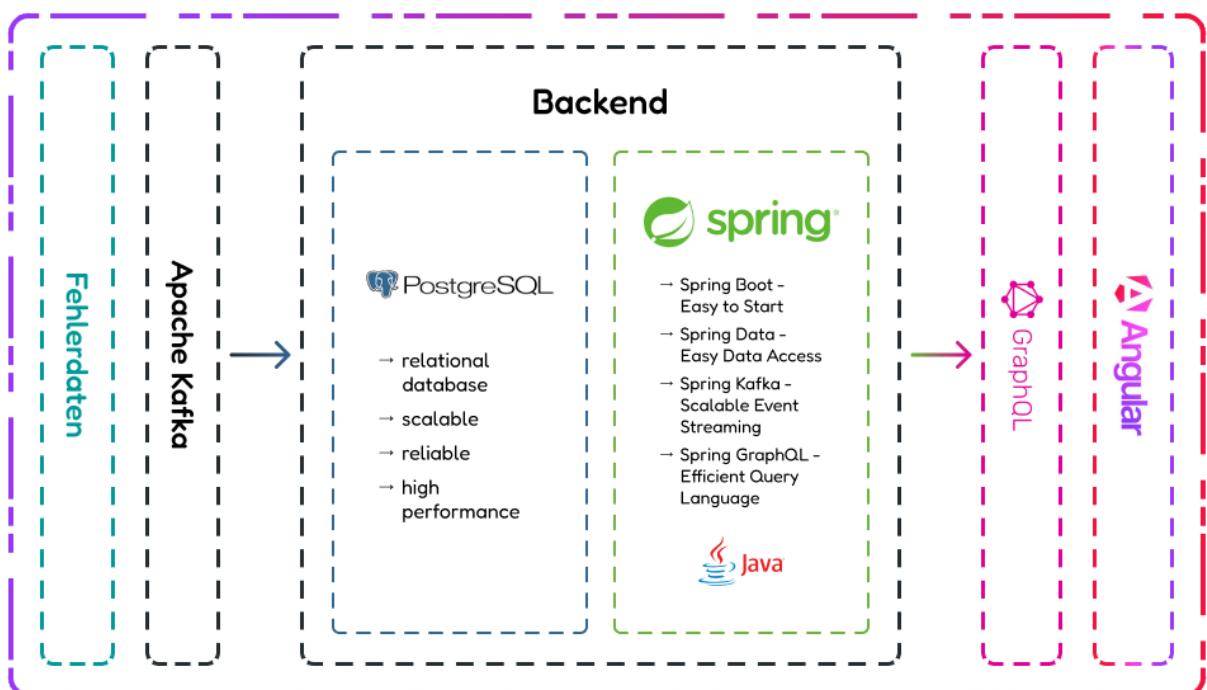


Abbildung 6.3.: Architektur unseres Prototypen mit Fokus auf das *Backend*

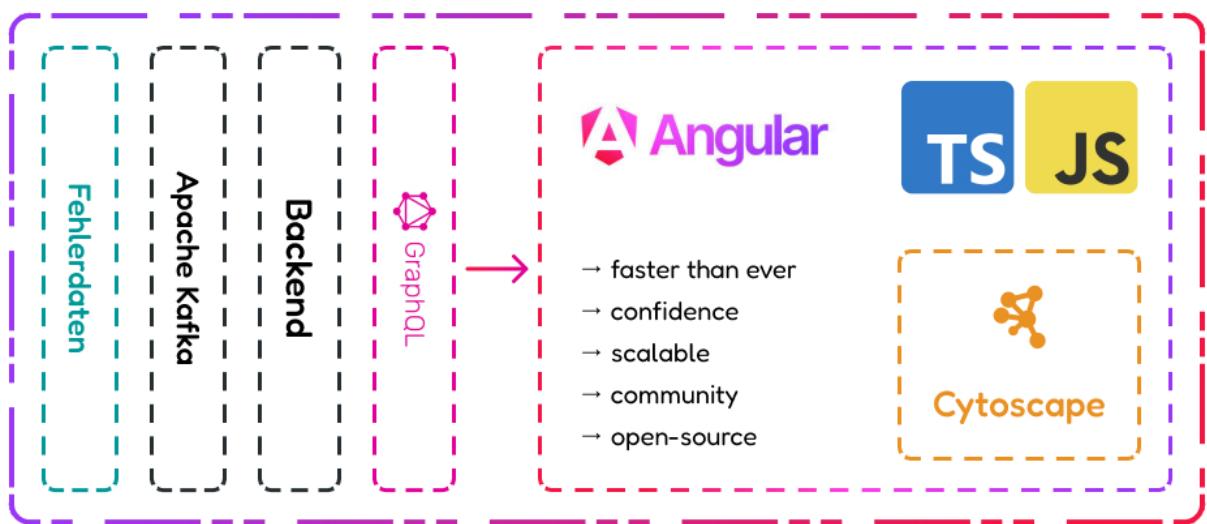


Abbildung 6.4.: Architektur unseres Prototypen mit Fokus auf das *Frontend*

7. Aufsetzen eines Kafka-Systems

7.1. Herausforderungen einer GraphQL API mit relationalen Datenbanken

8. Webapplikation

Angular ist ein von Google entwickeltes *Framework* zur einfachen Erstellung von verlässlichen 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.

Die Funktionalität der Komponenten bringt Struktur in die Applikation. Jede Komponente ist ein Teilstück Programmcode, bestehend aus *HTML*, *CSS* und *TypeScript*, welche eine in sich geschlossene Logik hat. Komponenten können jedoch auch Daten untereinander austauschen. Beispielsweise kann eine „TodoApp“ aus den Komponenten „TodoList“ und „TodoMetrics“ bestehen, wobei die Todo-Liste eine Kind-Komponente beinhaltet, „TodoListItem“. Hierbei ist die Kommunikation zwischen Liste und den einzelnen Elementen von großer Bedeutung, um die Abgeschlossenheit - ein Todo ist erfüllt - eines Elements richtig zu verwalten und anzuzeigen. [49]

Alle Komponenten werden in *Angular* mittels Baumstruktur gespeichert. In Abbildung 8.1 kann der Baum des obigen Beispiels betrachtet werden.

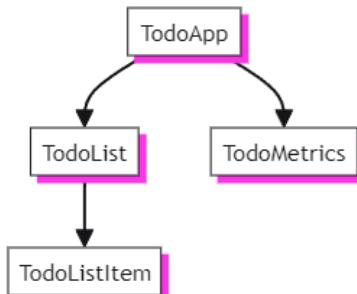


Abbildung 8.1.: Aufbau des Komponenten-Baumes in einer TodoApp [49]

In unserer Applikation verwende ich die mit *Angular*-Version 14 neu hinzugefügten *Standalone*-Komponenten. Diese Art von Komponenten erleichtern einerseits die Erstellung eines neuen Komponenten - dies ist aufgrund der *Angular CLI* (<https://angular.dev/cli>) sowieso schon äußerst einfach - und andererseits den Einstieg in das Entwickeln von *Angular*-Applikationen, da Programmierer das Konzept der *NgModules* nicht mehr erlernen müssen. [50]

In den frühen Stunden dieser Arbeit war das Hantieren der *Standalone*-Komponenten jedoch eine Herausforderung für mich, weil die modernisierte Dokumentation, welcher gleichzeitig mit *Angular* 17 auf den Markt kam (November 2023), anfangs Oktober noch nicht zur Verfügung stand. Aufgrund dieser Dokumentation fiel die Entscheidung, auf *Angular* 17 upzudaten einfacher, weshalb ich das Projekt mehrere Male neu aufsetzen musste.

Komponenten können nun mithilfe des zugehörigen *Selectors* innerhalb anderer Komponenten angezeigt werden. Außerdem bietet *Angular* die Möglichkeit, verschiedene „Routen“ zu verwenden, um in der *Single Page Applikation* unterschiedliche Inhalte bei unterschiedlichen URLs anzuzeigen.

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 *Dashboardes* 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, 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.

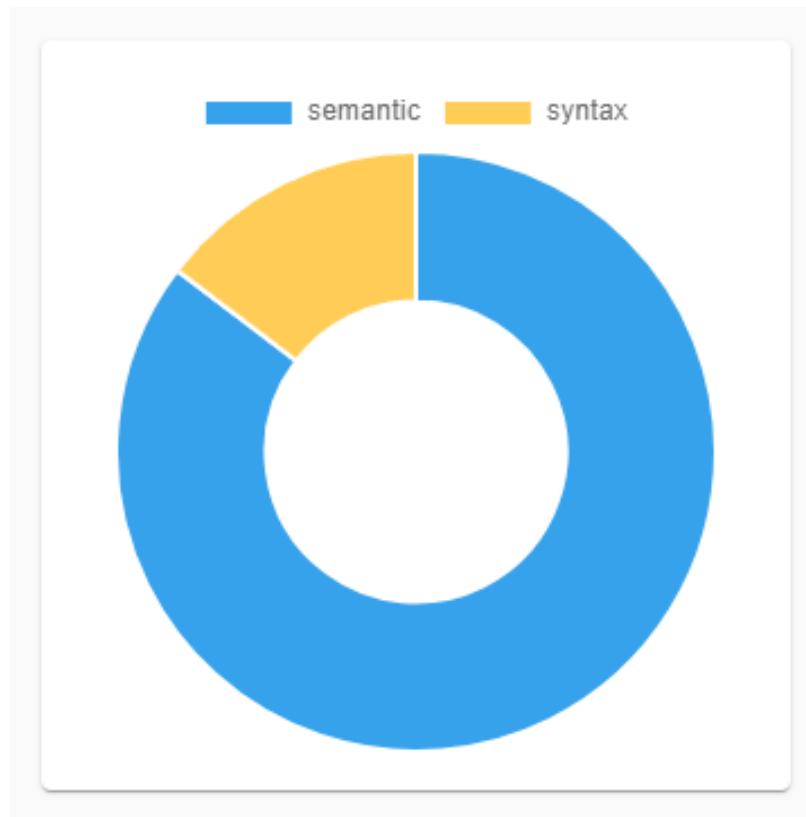


Abbildung 8.2.: Donatdiagramm zur Darstellung der prozentuellen Verteilung der Kategorie eines Fehlers im Netzmodell

8.2. Tabellen mit Filter- und Suchfunktionen

8.3. Graph des Stromnetzmodells

8.3.1. Einfache Integration von Cytoscape in Angular

Die Umsetzung der Visualisierung eines Graphen in *Angular* ist mittels *Cytoscape* [51] relativ einfach. Diese Bibliothek, welche normalerweise für biologische Darstellungen verwendet wird, bietet außerordentlich viele Möglichkeiten, Knoten und Kanten im Graphen zu erstellen, hinzuzufügen, zu löschen und ist mit einigen anderen *JavaScript*-Bibliotheken kompatibel.

Die Integration in *Angular* ist dank dem *Package Manager* npm recht einfach. Zuerst muss das Paket cytoscape installiert werden.

```
1   npm install cytoscape
```

Quellcode 8.1: *Cytoscape* installieren

Anschließend müssen die in *Cytoscape* definierten Typen in den ‚@types‘ Ordner gespeichert werden, was mit diesem Befehl erzielt werden kann:

```
1   npm i —save-dev @types/cytoscape
```

Quellcode 8.2: *Cytoscape* Typen speichern

Somit kann man bei jeder Komponente in *Angular* die Bibliothek importieren:

```
1   import cytoscape from 'cytoscape';
```

Quellcode 8.3: *Cytoscape* importieren

Cytoscape bietet nun Möglichkeiten, um einen Graphen zu erstellen und in ein bestimmtes „DOM“-Element zu rendern. Ein einfacher Graph könnte auf *Code*-Ebene beispielsweise so aussehen:

```

1   var cy = cytoscape({
2       container: document.getElementById('cy'), // container to render
3       in
4       elements: [
5           // list of graph elements to start with
6           {
7               // node a
8               data: { id: 'a' },
9           },
10          {
11              // node b
12              data: { id: 'b' },
13          },
14          {
15              // edge ab

```

```

16         data: { id: 'ab', source: 'a', target: 'b' },
17     },
18 ],
19
20     style: [
21         // the stylesheet for the graph
22         {
23             selector: 'node',
24             style: {
25                 'background-color': '#666',
26                 label: 'data(id)',
27             },
28         },
29     },
30     {
31         selector: 'edge',
32         style: {
33             width: 3,
34             'line-color': '#ccc',
35             'target-arrow-color': '#ccc',
36             'target-arrow-shape': 'triangle',
37             'curve-style': 'bezier',
38         },
39     },
40 ],
41
42     layout: {
43         name: 'grid',
44         rows: 1
45     },
46 });

```

Quellcode 8.4: einfachen Graph initialisieren

Hierbei muss man bei *Angular* darauf achten, dass der Graph erst initialisiert werden darf, wenn das DOM-Element gerendert wurde. Um sicherzustellen, dass dieser Rendervorgang bereits geschehen ist, gibt es die Funktion `ngAfterViewInit()`:

```

1  export class GraphComponent {
2      ngAfterViewInit(): void {
3          var cy = cytoscape({ ... });
4      }
5  }

```

Quellcode 8.5: Funktion `ngAfterViewInit`

Mit diesen Konfigurationen rendert *Angular* mittels *Cytoscape* einen Graphen, welcher in etwa so aussieht:

Cytoscape bietet verschiedene *Layouts*, sprich Anordnungen von Knoten und Kanten im *Container*, Animationen und stilistische Anpassungen und Adaptionen für den Graphen. Aktuelle API-Definitionen sind unter <https://js.cytoscape.org/> dokumentiert.



Abbildung 8.3.: einfacher *Cytoscape* Graph

8.3.2. Verwendung von Erweiterungen der Cytoscape-Bibliothek

Cytoscape bietet eine vielzahl an eigenen und *third-party* Erweiterungen, darunter auch einige *Graphlayouts*, welche auf Kraft- und Gewichtberechnungen basieren (D3, Cola, ...). Um dieses Erweiterungen in *Angular* einzubinden, muss man ein paar Schritte durchführen:

1. Paket installieren

```
1      npm install cytoscape-cola
2
```

Quellcode 8.6: *Cytoscape* installieren

2. Eine Datei erstellen, damit die Deklaration des Moduls gefunden werden kann

```
1      declare module 'cytoscape-cola';
2
```

Quellcode 8.7: *Cytoscape* installieren

3. Die Erweiterung importieren

```
1      import cola from 'cytoscape-cola';
2
```

Quellcode 8.8: *Cytoscape* installieren

4. Das Paket zu *Cytoscape* hinzufügen

```
1      cytoscape . use( cola );
2
```

Quellcode 8.9: *Cytoscape* installieren

Mithilfe der selbst definierten Erweiterungen, welche notwendig sind, um den Graph richtig rendern und *Angular* fehlerfrei laufen zu können, kann ich diese externen Bibliotheken anschließend verwenden.

```
1      import { BaseLayoutOptions } from 'cytosape';
2
3      export interface ColaLayoutOptions extends BaseLayoutOptions {
4          infinite?: boolean;
```

```
5      animate?: boolean;
6      refresh?: number;
7      ...
8  }
9
10 export const colaLayout = {
11   name: 'cola',
12   infinite: true,
13   animate: true,
14   refresh: 1,
15   ...
16 } as ColaLayoutOptions;
17
18 export interface D3LayoutOptions extends BaseLayoutOptions {
19   infinite?: boolean;
20   animate?: boolean | 'end';
21   maxIterations?: number;
22   ...
23 }
24
25 export const d3Layout = {
26   name: 'd3-force',
27   infinite: true,
28   maxSimulationTime: 69420,
29   ...
30 } as D3LayoutOptions;
```

Quellcode 8.10: Funktion ngAfterViewInit

```
1 import { colaLayout } from '@app/_models';
2
3 this.cy = cytoscape({
4   container: document.getElementById('cy'), // container to render
5   in
6   elements: this.nodesEdges as GraphElement[],
7   style: this.stylesheet as Stylesheet[],
8   layout: colaLayout,
9 })
10
11});
```

Quellcode 8.11: Funktion ngAfterViewInit

9. Bewertung

10. Zusammenfassung und Ausblick

10.1. Zusammenfassung

Zusammenfassend war diese Diplomarbeit ein sehr lehrreiches Projekt, bei dem wir viele neue Erfahrungen gemacht haben. ...

10.2. Ausblick

I. Literaturverzeichnis

- [1] 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.
- [2] 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.
- [3] Toshev, Martin: *Learning RabbitMQ: Build and Optimize Efficient Messaging Applications with Ease*. Packt Publishing, Birmingham, 2016, ISBN 978-1-78398-456-5.
- [4] Menge, Falko: *Enterprise Service Bus*. In: *Free and Open Source Software Conference*, Band 2, Seiten 1–6, 2007.
- [5] Narkhede, Neha: *Kafka: The Definitive Guide*. O'Reilly Media, 2017.
- [6] Fu, Guo, Yanfeng Zhang und Ge Yu: *A Fair Comparison of Message Queuing Systems*. IEEE Access, 9:421–432, 2021, ISSN 2169-3536.
- [7] Stopford, Ben: *Designing Event-Driven Systems*. 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*. 2018. Onlinequelle: <https://eplus.uni-salzburg.at/obvusbhs/content/titleinfo/4981548/full.pdf>.
- [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. Onlinequelle: <https://www.thi.uni-hannover.de/fileadmin/thi/publikationen/re-vo01.pdf>.
- [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. Onlinequelle: https://cognit.ch/michaelrichter/literat_97.pdf.
- [18] Krug, Steve: *Don't make me think!: Web & Mobile Usability: Das intuitive Web*. MITP-Verlags GmbH & Co. KG, 2018. Onlinequelle: https://books.google.at/books?id=e-VIDwAAQBAJ&printsec=frontcover&hl=de&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false.

- [19] Mardita, Rizki: *Designing Intuitive User Interface*. Medium, Mai 2017. Onlinequelle: <https://uxplanet.org/create-visual-ui-design-for-better-products-14f91e557638>.
- [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.
- [22] Ediger, Natalie: *What is Interactive Content?* cleverclip, 2020.
- [23] Neuberger, Christoph: *Interaktivität, Interaktion, Internet*. Publizistik, 52(1):33–50, 2007.
- [24] Morville, Peter und Jeffery Callender: *Search patterns: design for discovery*. O'Reilly Media, Inc., 2010. Onlinequelle: https://books.google.at/books?id=LzgNHuKKGJIC&printsec=frontcover&hl=de&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false.
- [25] 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.
- [26] 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.
- [27] Souders, Steve: *High-performance web sites*. Communications of the ACM, 51(12):36–41, 2008. Onlinequelle: <https://dl.acm.org/doi/fullHtml/10.1145/1409360.1409374>.
- [28] Sharma, Sushil und Pietro Murano: *A usability evaluation of Web user interface scrolling types*. First Monday, 2020. Onlinequelle: <https://firstmonday.org/ojs/index.php/fm/article/view/10309/9400>.
- [29] Hogan, Lara Callender: *Designing for Performance: Weighing Aesthetics and Speed*. O'Reilly Media, Inc., 2014. Onlinequelle: https://books.google.at/books?id=QPixBQAAQBAJ&printsec=frontcover&hl=de&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false.
- [30] Jorgensen, Corinne und Elizabeth D Liddy: *Information access or information anxiety?—An exploratory evaluation of book index*. Indexer, 1(20):64–68, 1996. Onlinequelle: <https://citeserxx.ist.psu.edu/document?repid=rep1&type=pdf&doi=44fb27a9e274e4975a6bea77af8c0542c3c9600b>.
- [31] 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.
- [32] statcounter. Onlinequelle: <https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet>.
- [33] Fry, Ben: *Visualizing data*. O'Reilly Media, Inc., 2008. Onlinequelle: https://books.google.at/books?id=RRswXg4pJhcC&printsec=frontcover&hl=de&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false.
- [34] *The Top 10 Types of Data Visualization Made Simple*. Boost Labs, 2020.

- [35] Morales, José Miguel: *What is Data Visualization? From Data to Visualization.* BIUWER, 2020.
- [36] French, Katie: *25 Tips to Instantly Improve Your Data Visualization Design.* Column Five.
- [37] *Welches Diagramm für welche Daten? So finden Sie die richtige Diagrammtypen.* insightsoftware, 2023.
- [38] Marktler, Josef: *12 coole Typen – Wer ist der Richtige?* storytellingmitdaten, 2020.
- [39] Zeising, Tobias: *Was lesen Buchblogger: Eine neue Analyse mit Visualisierungen und Statistiken.* lesestunden, 2016.
- [40] Lin, Pohan: *2023 Guide to Big Data Visualization.* Piktochart, 2023.
- [41] Heins, Eberhard: *Daten-Tools visualisieren Big-Data-Potenzial.* the IT-Matchmaker, 2017.
- [42] Wampfler, Philippe: *Methodische Probleme von Big Data im Umgang mit Social Media,* 2014.
- [43] Pusic, Ante: *Graph Clustering Algorithms: Usage and Comparison.* memgraph, 2023.
- [44] Yi, Mike: *A complete guide to heatmaps.* Atlassian.
- [45] Cinar, Özlem: *Visualisierung der Ergebnisse (MOS) Heatmap.* 2023.
- [46] Kassamara, Alboukadel: *Hierarchical Clustering in R: The Essentials Heatmap in R: Static and Interactive Visualization.* DataNovia, 2020.
- [47] Bostock, Mike: *Fisheye Distortion.* 2012.
- [48] Beilhammer, Marius: *Datenvisualisierung: Dateninterpretation auf einen Blick.* industryPRESS, 2017.
- [49] Die Dokumentation wurde mit Angular Version 17 erneuert und modernisiert.
- [50] Ahmed, Nisar: *What Are Standalone Components and How to Utilize Them in Angular?* Medium, 2023.
- [51] *Cytoscape JavaScript Library.* Oxford Bioinformatics (2016, 2023). Onlinequelle: <https://js.cytoscape.org/>.
- [52] Freeman, Linton C.: *Visualizing Social Networks.* University of California, Irvine, 2000. Onlinequelle: <https://bebr.ufl.edu/sites/default/files/Freeman%20-%202000%20-%20Visualizing%20social%20networks.pdf>.
- [53] Ware, Colin: *Information Visualization: Perception for Design.* Elsevier Inc., 2021. Onlinequelle: https://books.google.at/books?id=3-HFDwAAQBAJ&printsec=frontcover&hl=de&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false.
- [54] 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. Onlinequelle: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=cdecf712eb5bd499ed2c2fd52bfc57d0b22e24b4>.

- [55] 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. Onlinequelle: https://books.google.at/books?id=i0NsCQAAQBAJ&printsec=frontcover&hl=de&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false.
- [56] Jünger, Michael und Petra Mutzel: *Graph drawing software*. Springer Science & Business Media, 2012.
- [57] Wright, Chris: *Web Optimization: The Myth of the 3 Click Rule*. CMS-Wire, 2010. Onlinequelle: <https://www.cmswire.com/cms/web-engagement/web-optimization-the-myth-of-the-3-click-rule-009018.php>.
- [58] Laubheimer, Page: *The 3-Click Rule for Navigation Is False*. Nielsen Norman Group, 2019. Onlinequelle: <https://www.nngroup.com/articles/3-click-rule/>.
- [59] *Pagination Examples that Work – We Analyzed the Most Effective Strategies*, 2023.
- [60] Kim, Jaewon, Paul Thomas, Ramesh Sankaranarayana, Tom Gedeon und Hwan Jin Yoon: *Pagination versus scrolling in mobile web search*. In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, Seiten 751–760, 2016.
- [61] Doin: *Logarithmic Page Navigation*, 2013.
- [62] Kepner, Jeremy, David Bader, Aydin Buluç, John Gilbert, Timothy Mattson und Henning Meyerhenke: *Graphs, matrices, and the GraphBLAS: Seven good reasons*. Procedia Computer Science, 51:2453–2462, 2015.
- [63] Bavdekar, Sandeep B: *Using tables and graphs for reporting data*. J Assoc Physicians India, 63(10):59–63, 2015. In dieser Arbeit werden die Begriffe 'Graph' und 'Diagramm' synonym verwendet, um auf grafische Darstellungen von Daten oder Beziehungen hinzuweisen, ohne dabei auf etwaige fachliche Unterschiede einzugehen.
- [64] Paralogical: *I removed most of the syllables from english and it's 30% faster now*. YouTube, 2023. Onlinequelle: <https://www.youtube.com/watch?v=sRbcw2sGkJw>.
- [65] Rodgers, Peter, Gem Stapleton und Peter Chapman: *Visualizing sets with linear diagrams*. ACM Transactions on Computer-Human Interaction (TOCHI), 22(6):1–39, 2015.
- [66] Gutwenger, Carsten, Michael Jünger, Karsten Klein, Joachim Kupke, Sebastian Leipert und Petra Mutzel: *A new approach for visualizing UML class diagrams*. In: *Proceedings of the 2003 ACM symposium on Software visualization*, Seiten 179–188, 2003.
- [67] Jacobsen, Jens: *Daten, Diagramme, Dashboards gute UX*. Usabilityblog, 2017.
- [68] Liu, Jia Xin: *Challenges of increasing usability on table filter for huge amounts of data*, 2022.
- [69] Campos, Marcelo, Simon Griffiths, Robert Morris und Julian Sahasrabudhe: *An exponential improvement for diagonal Ramsey*. arXiv preprint arXiv:2303.09521, 2023. Onlinequelle: <https://arxiv.org/pdf/2303.09521.pdf>.
- [70] Moraru, Andrei Lucian: *The Problem With Too Many JavaScript Frameworks*. Medium, 2021. Onlinequelle: <https://betterprogramming.pub/the-problem-with-too-many-js-frameworks-11531ac8b896>.

- [71] *Spline*. Onlinequelle: <https://spline.design/>.
- [72] *D3 JavaScript Library*. Onlinequelle: <https://d3js.org/>.
- [73] Shannon, Paul, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski und Trey Ideker: *Cytoscape: a software environment for integrated models of biomolecular interaction networks*. Genome research, 13(11):2498–2504, 2003.
- [74] Morgenstern, U. und R. Freyer: *WERKZEUGE UND METHODIK FÜR MULTIMEDIALE LERNSYSTEME IN BMT UND MEDIZIN – ein Beispiel zur mehrdimensionalen Datenerfassung, -verarbeitung und -visualisierung*. Biomedical Engineering / Biomedizinische Technik, 45(s1):21–22, 2000. <https://doi.org/10.1515/bmte.2000.45.s1.21>.
- [75] Keim, Daniel, Huamin Qu und Kwan Liu Ma: *Big-data visualization*. IEEE computer graphics and applications, 33(4):20–21, 2013.
- [76] Olavsrud, Thor: *Was ist Datenvisualisierung?*, 2021.
- [77]

II. Abbildungsverzeichnis

3.1. Ein Beispiel für ein verteiltes System mit direkter Kommunikation [1]	16
3.2. System mit einer Datenbank als zentrale Kommunikationskomponente [3]	17
3.3. Ein Beispiel für ein verteiltes System mit einem zentralen <i>Messaging System</i> [1]	18
3.4. Ein einfacher <i>Enterprise Service Bus</i> [4]	19
3.5. Eine <i>Queue</i> [1]	20
3.6. Das <i>Point-to-Point Messaging Model</i> [1]	21
3.7. Das <i>Publish-Subscribe Messaging Model</i> [1]	22
3.8. Eine <i>consumer group</i> verarbeitet gemeinsam ein Topic. [5]	26
3.9. Die Architektur des Apache Kafka Systems [6]	26
3.10. Die Architektur des RabbitMQ Systems [6]	28
4.1. Zwei Graphen dessen Unterschied die Ausrichtung ist	29
4.2. Ein Graph all dessen Knoten verbunden sind (links); ein Graph, welcher nicht verbundene Knoten enthält (rechts)	30
4.3. Ein Graph, welcher keinen geschlossenen Kreis (Zyklus) enthält (links); ein Graph mit Zyklus (rechts)	31
4.4. Ein Graph, welcher gewichtete Kanten hat (links); ein Graph, dessen Kanten immer das gleiche Gewicht haben (rechts)	31
4.5. Ein Graph, wo jeder Knoten mit allen anderen Knoten verbunden ist (links); nicht alle Knoten sind miteinander verbunden (rechts)	32
4.6. Graph, welcher zwei Mengen ohne inhärente Verbindungen hat (links); keine Bipartition möglich beim rechten Graph	32
4.7. Ein Graph ohne Schnittpunkte der Kanten, wenn alle Knoten verbunden sind (links); nicht möglich ohne Schnittpunkte (rechts)	33
4.8. Graph mit einem Zyklus, welcher alle Kanten einmal enthält (links); Graph ohne diesen Eulerkreis	33
4.9. Graph mit Zyklus, welcher alle Knoten beinhaltet (links); kein Hamiltonscher Kreis möglich (rechts)	34
5.1. Einfache Datenvizualisierungen in Form von Werten, Diagrammen, Tabellen und Karten [35]	38
5.2. Beliebtheit der verschiedenen Buchgenre im Jahr 2016 [39]	39
5.3. Stark vernetzte Daten in einem Graphen (LinkedIn Netzwerk) [42]	40
5.4. <i>Clustering</i> eines Graphen [43]	41
5.5. <i>Heatmap</i> der Beiträge eines GitHub Benutzers im Jahr 2022 [44]	42
5.6. Demonstration der Lupentechnik „ <i>Fisheye</i> “ auf einem Grid [47]	43
5.7. Demonstration des Fischauges in einem Graphen [Source: GitHub Gist]	43
5.8. Demonstration der Lupentechnik „ <i>Kartesisch</i> “ auf einem Grid [47]	43
5.9. Exemplarische geobasierte <i>Heatmap</i> aus der Google Maps Dokumentation	44
5.10. Architektur unseres Prototypen mit Fokus auf die Bibliothek <i>Cytoscape</i>	45
6.1. Übersicht der Architektur unseres Prototypen	46
6.2. Architektur unseres Prototypen mit Fokus auf Kafka	47
6.3. Architektur unseres Prototypen mit Fokus auf das <i>Backend</i>	47
6.4. Architektur unseres Prototypen mit Fokus auf das <i>Frontend</i>	48
8.1. Aufbau des Komponenten-Baumes in einer TodoApp [49]	50

8.2. Donatdiagramm zur Darstellung der prozentuellen Verteilung der Kategorie eines Fehlers im Netzmodell	51
8.3. einfacher <i>Cytoscape</i> Graph	54
A.1. Jira Plan	93

III. Tabellenverzeichnis

A.1. Kapitelverzeichnis	69
A.2. Arbeitstagebuch Schlipfinger	94
A.3. Arbeitstagebuch Schneider	96

IV. Quellcodeverzeichnis

8.1.	<i>Cytoscape</i> installieren	52
8.2.	<i>Cytoscape</i> Typen speichern	52
8.3.	<i>Cytoscape</i> importieren	52
8.4.	einfachen Graph initialisieren	52
8.5.	Funktion ngAfterViewInit	53
8.6.	<i>Cytoscape</i> installieren	54
8.7.	<i>Cytoscape</i> installieren	54
8.8.	<i>Cytoscape</i> installieren	54
8.9.	<i>Cytoscape</i> installieren	54
8.10.	Funktion ngAfterViewInit	54
8.11.	Funktion ngAfterViewInit	55

htlkrems	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

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 Softwarepaketes; Teil einer Applikation

A. Anhang

A.1. Kapitelverzeichnis

Kapitel	Autor
1. Präambel	Schneider Felix
2. Einleitung	Schneider Felix
3. Message Propagation	Schlipfinger Clemens
4. Graphentheorie	Schneider Felix
5. Visualisierung	Schneider Felix
6 Architektur	Schneider Felix
6.1 Backend	Schlipfinger Clemens
6.2 Frontend	Schneider Felix
7. Kafka	Schlipfinger Clemens
8. API	Schlipfinger Clemens
9. Webapplikation	Schneider Felix
10.1 Bewertung des Enterprise Service Bus Systems	Schlipfinger Clemens
10.2 Bewertung der Visualisierungsmethoden für stark vernetzte Daten	Schneider Felix
11. Zusammenfassung	Schlipfinger Clemens
12. 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 Schlipfinger Backend programmer

Felix Schneider Frontend programmer

Version History

Version	Datum	AutorIn	Änderungen
0.1	28.10.2023	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: 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

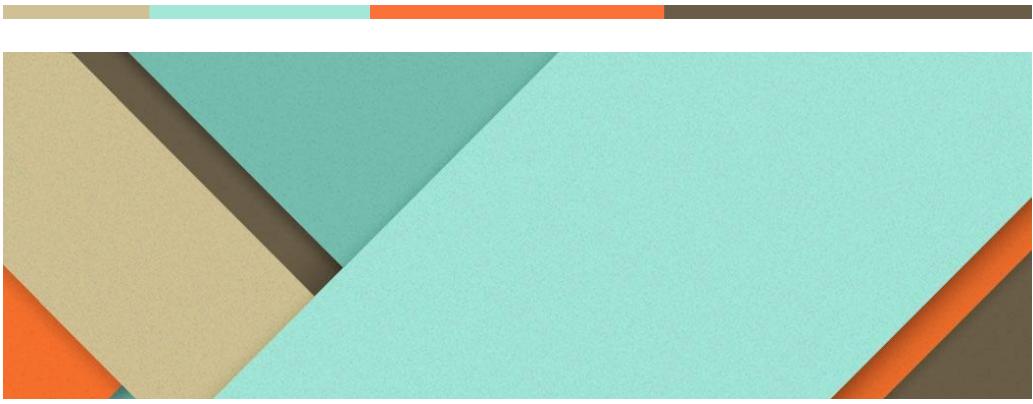
- Pflichtenheft
 - Datum bei Suche?
 - Deployment/Abgabe als Docker-Container?
 - Mockup:
 - Dashboard mehr Content
 - Accounts müssen weg!
 - Filter: Sortierung
 - Zeichnung der Systemarchitektur mit Farben und Icons erweitern.

Next Steps

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

htlkrems	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
Abteilung:	Informationstechnologie

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 Schlipfinger Backend programmer
 Felix Schneider Frontend programmer

Version History

Version	Datum	AutorIn	Änderungen
0.1	13.12.2023	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: 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

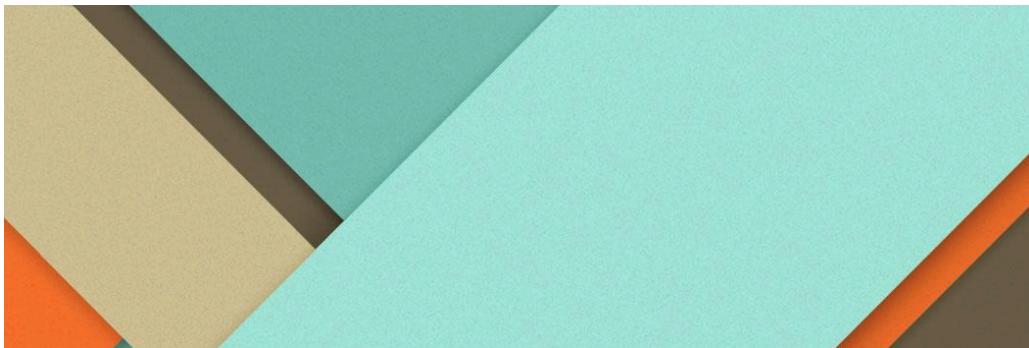
- Theoretischen Teil
- Jira Plan anschauen
 - Nicht erledigte Issues (mitten im Sprint hinzugefügt) zurück in Backlog
- Empirischen Teil
 - Zuerst Architektur beschreiben, dort kommen dann neue Themen, zB GraphQL, welche kurz beschrieben werden können
 - Interessante / schwierige Themen, welche während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

Meeting Minutes / Begleitprotokoll 3

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 Schlipfinger Backend programmer

Felix Schneider Frontend programmer

Version History

Version	Datum	AutorIn	Änderungen
0.1	13.02.2024	Felix Schneider & Clemens Schlipfinger	Erstellung des Dokuments; Meeting verschieben aufgrund Zeitmangel
0.2	16.02.2024	Felix Schneider & Clemens Schlipfinger	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 reinplatzieren.
 - 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

width=!,height=!,pages=1,scale=.85,pagecommand=

A.2.5. Begleitprotokoll 5

width=!,height=!,pages=2-,scale=.85,pagecommand=

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.

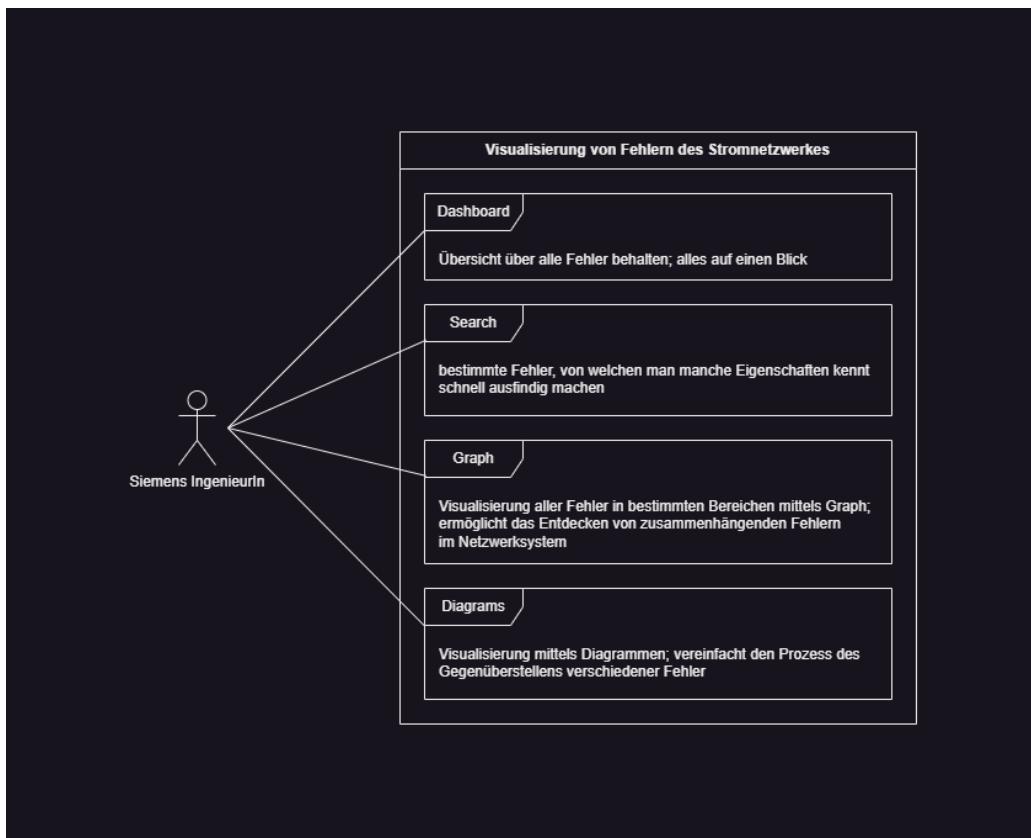
- Es müssen die Daten mit Hilfe einer Datenbank **persistiert** werden und bis zur nächsten Netzmodellfehleranalyse gehalten werden.
- Die Findings müssen durch eine **Schnittstelle (GraphQL) dem Frontend** angeboten werden.
- Der Backend soll sich als **optionale Erweiterung** leicht in das bisherige System integrieren lassen. Zusätzlich soll es nur kleine Veränderungen am Haupt-System erzwingen.
 - Für den Entwicklungsprozess soll eine **Simulationsssoftware** 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

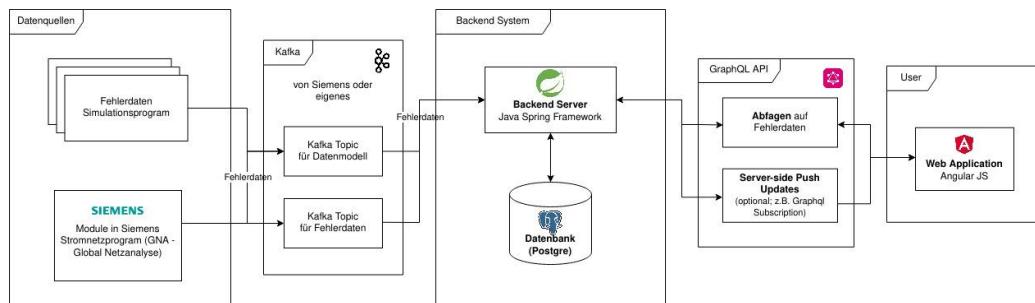
[Non-Functional Requirements]

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
Barriere-freiheit	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:



	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems	
Abteilung:	Informationstechnologie	

A.4. Arbeitspakte

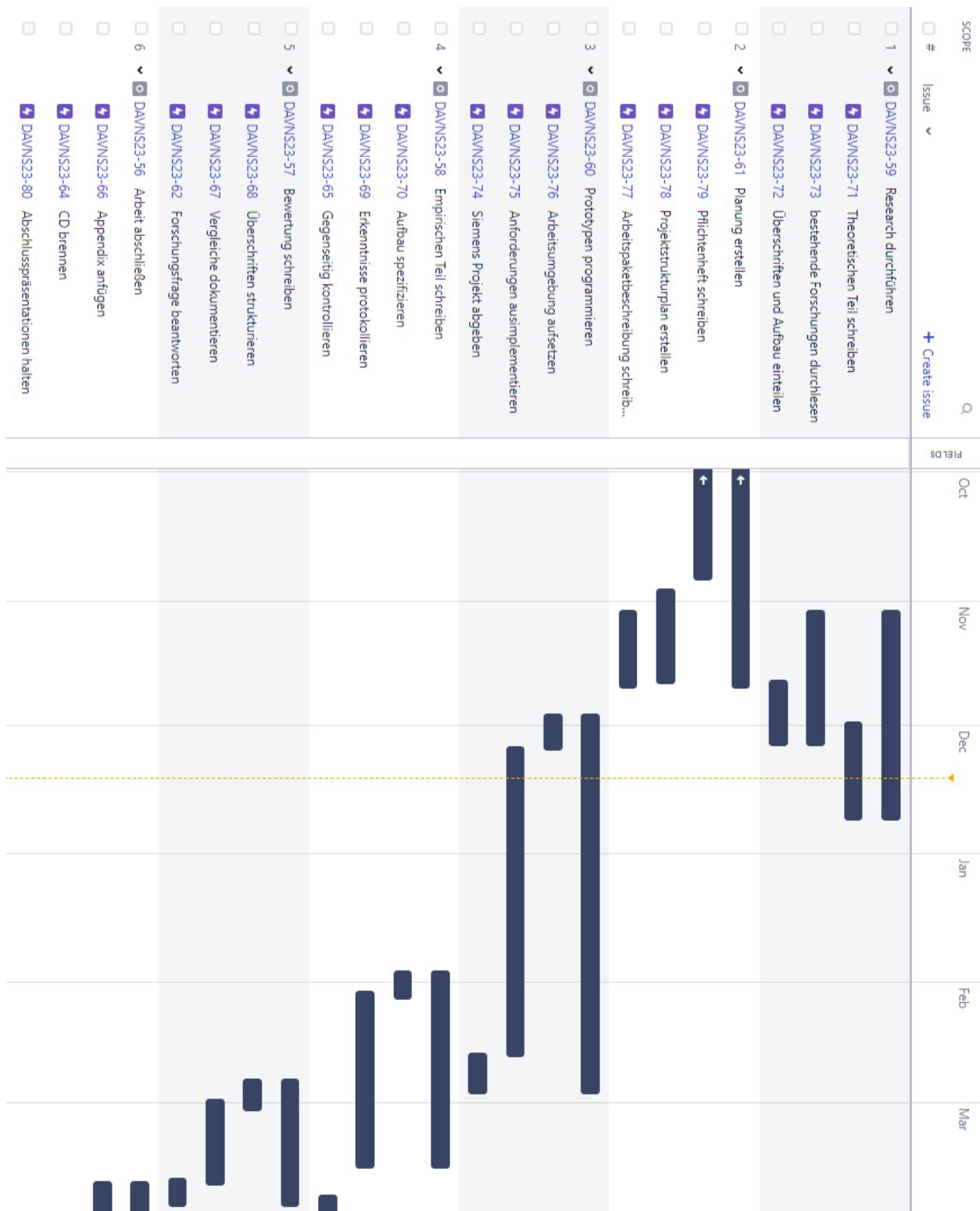


Abbildung A.1.: Jira Plan

A.5. Projekttagebücher

A.5.1. Projekttagebuch Clemens Schlipfinger

Tag	Zeit	kumulativ	Fortschritt
-----	------	-----------	-------------

Tabelle A.2.: Arbeitstagebuch Schlipfinger

A.5.2. Projekttagebuch 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	00:58:00	05:09:15	Mockup Figma View erstellen
21 September	00:32:00	05:41:15	Mockup Figma View erstellen
21 September	01:43:00	07:24:15	Mockup Figma View erstellen
23 September	02:30:00	09:54:15	Diplomarbeit Datenbank Felder ausfüllen
10 Oktober	00:18:00	10:12:15	UseCase Frontend für Pflichtenheft
13 Oktober	03:00:00	13:12:15	Research Graph Visualisation
19 Oktober	01:52:00	15:04:15	Mockup
19 Oktober	01:21:00	16:25:15	Mockup
19 Oktober	01:03:47	17:29:02	Mockup
23 Oktober	00:17:46	17:46:48	Mockup
23 Oktober	00:20:25	18:07:13	Mockup
24 Oktober	00:37:49	18:45:02	Mockup
24 Oktober	00:02:04	18:47:06	Mockup
25 Oktober	01:27:02	20:14:08	Mockup
25 Oktober	00:53:00	21:07:08	Mockup
25 Oktober	00:27:04	21:34:12	Pflichtenheft
25 Oktober	00:10:23	21:44:35	Mockup
25 Oktober	00:01:12	21:45:47	Pflichtenheft
26 Oktober	00:35:09	22:20:56	Pflichtenheft
26 Oktober	00:05:34	22:26:33	Mockup
27 Oktober	00:51:59	23:18:32	Pflichtenheft
28 Oktober	01:20:00	24:38:32	Besprechung
29 Oktober	00:30:00	25:08:32	Mockup
29 Oktober	00:23:14	25:31:46	Mockup
30 Oktober	06:00:00	31:31:46	Meeting bei Siemens; Überarbeitung Pflichtenheft; Kleinigkeiten klären
01 November	00:30:00	32:01:46	Jira Plan erstellen
01 November	00:30:00	32:31:46	Jira Plan erstellen
01 November	00:30:00	33:01:46	Confluence weiterarbeiten
03 November	00:45:00	33:46:46	Jira Plan und Confluence
03 November	00:21:42	34:08:28	Mockup
03 November	00:31:16	34:39:44	Mockup
03 November	02:06:28	36:46:12	Learn Angular
16 November	00:50:59	37:37:11	Jira Plan erstellen
16 November	00:21:11	37:58:22	Jira Plan erstellen
10 Dezember	00:20:38	38:19:00	Angular Table
10 Dezember	00:51:16	39:10:16	Grundstruktur aufbauen
13 Dezember	00:45:00	39:55:16	Meeting Minutes 2
13 Dezember	03:00:00	42:55:16	Datenbankstruktur Modellierung und Research Visualisierungsmethoden
14 Dezember	00:28:10	43:23:26	Research zitieren Benutzerfreundlichkeit Seite 96 von 97
16 Dezember	01:11:49	44:35:15	Research zitieren Benutzerfreundlichkeit

	HÖHERE TECHNISCHE BUNDES - LEHRANSTALT Krems
Abteilung:	Informationstechnologie

A.6. Datenträgerbeschreibung

Auf der beigelegten DVD befinden sich folgende Dateien:

- Die schriftliche Arbeit befindet sich unter **thesis/documentation/final.pdf**
- Die schriftliche Arbeit in L^AT_EX-Format befindet sich unter **thesis/latex/**
- Sämtliche Dateien des Projektmanagements befinden sich unter **thesis/management/**
- Zwei Docker Container für das *Backend* und das *Frontend* befinden sich unter **thesis/-docker/**
- Eine Datei zum Ausführen der gesamten Applikation befindet sich unter **thesis/run/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.