SENTIMENTS OF BUNDESTAG

Graph-basiertes Informationssystem zur Analyse sozialer Interaktion im Deutschen Bundestag

25. Februar 2021

Deutscher Bundestag

Betreut von Prof. Dr. Thomas Hoppe
Wahlpflichtmodul 4: IS Information Systems
M.Sc. Angewandte Informatik
Hochschule für Technik und Wirtschaft
Treskowallee 8, 10318 Berlin, Deutschland

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Vorwort

1.1 Einleitung

Einen Beitrag vom 13.11.2019 zur aktuellen Stimmung im Deutschen Bundestag betitelt die ZDF-Redaktion mit „Der raue Ton im Bundestag“ [1]. Sie ist der Ansicht, dass der Umgangston seit der Wahl der AfD in den Bundestag rauer und aggressiver geworden ist. Die Redaktion stützt sich dabei auf zahlreiche Reden, Reaktionen und Kommentare von Abgeordneten verschiedener Parteien. Die bei einer Bundestagssitzung besprochenen Inhalte und Reaktionen werden von Stenographen in Plenarprotokollen ausführlich dokumentiert. Diese Protokolle sind frei zugänglich auf der Webseite des Bundestages. Auch die Stammdaten, eine Auflistung aller Abgeordneter, die jemals im Bundestag vertreten waren (Open Data [2]), sind öffentlich. Im Rahmen des Moduls Information Systems im Master-Studiengang Angewandte Informatik der HTW Berlin und auf Vorschlag von Prof. Dr. rer. nat. Thomas Hoppe wurde ein Projekt gestartet, welches auf Basis der Plenarprotokolle die Aussage der ZDF-Redaktion prüfen möchte. Dies soll mithilfe eines graphbasierten Informationssystems, welches die sozialen Interaktionen im Deutschen Bundestag umfasst, gelingen. Aus den Plenarprotokollen sollen durch ein Kommunikationsmodell die Interaktionen ermittelt und anschließend die jeweilige Stimmung errechnet werden. Auf Basis dieser Daten sollen weitere Berechnungen für einzelne Abgeordnete oder ganze Frktionen angestellt und als ein Sentiment-Graph persistiert werden. Dieser Graph soll als Grundlage für eine ausführliche Visualisierung dienen. Zur Lösung dieser Aufgaben wurde das Projekt in sieben Teilprojekte unterteilt. Jede Gruppe, welche im Durchschnitt aus drei Studierenden bestand, hat sich mit der jeweiligen aufgetragenen Thematik beschäftigt und diese bearbeitet. Auf die Thematik der jeweiligen Teilprojekte sowie auf deren konkreten Zielsetzung und Umsetzung wird im nächsten Abschnitt genauer eingegangen.

1.2 Aufbau der Lösung

Das Projekt zur Entwicklung eines graphbasierten Informationssystem für die Analyse sozialer Interaktionen im Deutschen Bundestag, welches „Sentiments Of Bundestag“ genannt wurde, besteht, wie bereits erwähnt, aus sieben Teilprojekten. Im folgenden Abschnitt wird grob auf die Thematiken der einzelnen Gruppen eingegangen.

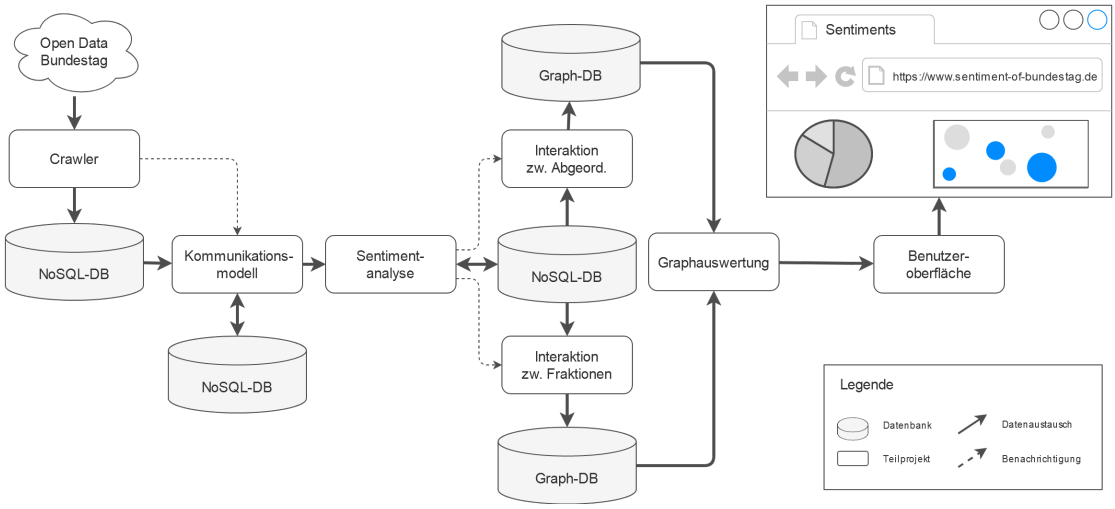

Abbildung 1.1: Aufbau der Lösung

Die in Abbildung 1.1 dargestellten Teilprojekte sind hier nun detaillierter aufgelistet:

- **Crawler**: Scannt regelmäßig die Open Data Webseite des Bundestags, sucht, parst und speichern neue Protokollen sowie Stammdaten der Abgeordneten in seiner No-Sql-Datenbank. Ziel ist es hier sicherzustellen, das die DB immer auf dem neusten Stand bleibt
- **Kommunikationsmodell**: Analysiert und erstellt aus den Protokollen von Gruppe 1 ein Kommunikationsmodell, welches die möglichen Interaktionen im Bundestag abbildet
- **Sentiment-Analyse**: Errechnet die Stimmung der von Gruppe 2 identifizierten Interaktionen und stellt Daten für Gruppe 3 und 4 bereit
- **Interaktion zwischen Abgeordneten**: Aus dem Kommunikationsmodell von Gruppe 2 und die Stimmungsanalyse von Gruppe 3 werden hier Interaktionen zwischen einzelnen Personen (Abgeordneten, Präsident, Gäste, etc.)

identifiziert. Erstellt wird wird daraus ein gewichteter Sentiment-Graph zwischen Abgeordneten mit positiven/negativen Gewichtungen

- **Interaktion zwischen Fraktionen**: Aus dem Kommunikationsmodell von Gruppe 2 und den Sentiment-Graph zwischen Personen von Gruppe 4 werden hier Interaktionen zwischen Gruppen von Personen analysiert und in einen Sentiment-Graph zwischen Parteien. Der besteht aus einer Aggregation der Abgeordnetensentiments zu gewichteten Sentiment-Graph der Parteien (Fraktionen, Gruppen, etc.)

- **Graphauswertung**: Die Sentiment-Graphen von den Gruppen 4 und 5 werden hier anhand verschiedener Auswertungsmethoden analysiert und die Ergebnisse davon der nächsten Gruppe (Benutzeroberfläche) zur Verfügung gestellt

- **Benutzeroberfläche**: Ziel ist hier die Realisierung einer interaktiven Benutzeroberfläche zur Darstellung der Ergebnisse

Die einzelnen Teilprojekte werden in den nächsten Kapiteln von den jeweiligen Gruppenmitgliedern genauer erläutert.

Kapitel 2

Crawler

Gruppenmitglieder: Arnauld Feussi, Boris Foko Kouti, Marlon Daniel Kohlberger

Die Umsetzung dieses Teilprojekt sowie die Erstellung dieser Dokumentation wurde wie in der nachfolgenden Tabelle angegeben aufgeteilt.

Tabelle 2.1: Gruppe 1 (Crawler) - Arbeitsaufteilung

Aufgabe	Gruppenmietglieder	Anteil
Crawl-Manager	Boris Foko Kouti	Scheduler, TaskManager, Rest-API
Crawl-Utilities	Marlon Daniel Kohlberger Boris Foko Kouti	XML-Parser Page Loader, Page Analyser
Datenbank	Marlon Daniel Kohlberger Boris Foko Kouti	DB-Modell DB-Manager
Dokumenation	Arnauld Feussi Marlon Daniel Kohlberger Boris Foko Kouti	Erste Vorlage DB-Modell, Parser, Refactoring Einleitung, Crawl-Mechanismus und Bereitstellung
Bereitstellung	Arnauld Feussi Marlon Daniel Kohlberger	Server Aufsetzung Sicherheit Config und Firewall Config
	Boris Foko Kouti	Crawler Service und MongoDB

2.1 Einleitung

Die Umsetzung eines graphbasierten Informationssystems zur Analyse sozialer Interaktionen im Deutschen Bundestag setzt voraus, dass aktuelle Informationen und Berichte über die Debatten im Bundestag zu jeder Zeit zur Verfügung stehen. Diese Informationen sowie Stammdaten von Abgeordneten und anderen Sitzungsteilnehmern werden regelmäßig auf der Seite des Bundestags veröffentlicht und sind für alle interessierten Anwender frei zugänglich [2]. Das Durchsuchen der Webseiten, das Vergleichen, Herunterladen und Parsen der Protokolle, als auch der Stammdaten, lässt sich mithilfe eines Computerprogramms automatisieren. Ein solches Programm wird als Crawler bezeichnet. Im Allgemeinen besteht die Aufgabe eines Crawlers darin sich wiederholende Operationen, wie zum Beispiel der Download oder die Indexierung einer Webseite, soweit wie möglich zu systematisieren und ohne manuellen Eingriff regelmäßig auszuführen. Ein Crawler funktioniert in der Regel wie ein Bot und kann zur Extraktion bestimmter Informationen von einer oder mehreren Webseiten verwendet werden. Des Weiteren kann dieser auch eine komplette ggf. leicht abgewandelte Replikation des Inhalts einer Webseite erstellen. Im zweiten Fall geht es um einen sogenannten Scraper. Zur Sammlung der vom graphbasierten Informationssystems benötigten Daten wird ein Crawler eingesetzt, der wie bei einem Scraper eine bestimmte Seite in einer festgelegten Frequenz herunterlädt und aus dieser Links für spezifische Datentypen extrahiert. Abhängig davon, ob die gesammelten Links bereits bekannt sind oder nicht, werden die entsprechenden Dateien gedownloadet, geparst und anschließend in einer Datenbank gespeichert. Bevor der Crawler implementiert wird, ist es zunächst wichtig genaue technische Anforderungen aufzustellen, sowie eventuelle Schwierigkeiten, in diesem Fall auch speziell für die Bundestag-Seite, zu identifizieren. Daraus wird ein Konzept zur Lösung dieser Aufgabe erarbeitet. Auf die beiden zuvor genannten Punkte, die Implementierung des vorgeschlagenen Konzepts und die Bereitstellung der vollständigen Lösung, wird in den folgenden Abschnitten dieses Dokuments näher eingegangen.

2.2 Anforderungen und Rahmenbedingungen

Der zu entwickelnde Crawler soll in bestimmten Rahmenbedingungen einige Anforderungen erfüllen. Die technischen als auch organisatorischen Anforderungen und Rahmenbedingungen werden in diesem Abschnitt aufgelistet.

2.2.1 Anforderungen an dem Crawler

Die Anforderungen an den Crawler sind in [3] zusammengefasst und sehen vor, dass der Crawler kontinuierlich laufen soll und dabei nach folgenden Dateien sucht:

- **Protokolle**: Ein Protokoll ist eine XML-Datei, die den gesamten Ablauf einer Plenarsitzung beinhaltet, so wie dessen Teilnehmer. Da sich die Formatierung der Protokolle in den Jahren stets verändert haben, soll sich hier nur auf die folgenden Legislaturperioden bezogen werden:
  - 19. Legislaturperiode: XML-Dokumente, welche somit gut strukturiert sind und das parsen erleichtern
  - 18. Legislaturperiode: Text-Datei, welche schlechter strukturiert ist als die 19. Legislaturperiode. Hierzu läuft derzeit ein Projekt, bei welchem die Protokolle mithilfe einer KI in XML-Dokumente umgewandelt werden, welche die selbe Form wie die der 19. Periode aufweisen. Nach Erfolg des genannten Projektes ist es möglich diese Protokolle mit dem selben Parser der 19. Legislatur zu parsen.
- **Stammdaten der Abgeordneten**: XML-Dokument, welches Informationen über alle Abgeordneten seit 1949 sammelt
- *Optional* **Namentliche Abstimmungen**: Liste aller namentlichen Abstimmungen im Bundestag seit dem 18.12.2009 in PDF-Format und seit dem 18.10.2012 auch in XLSX-Format [4]
- Es muss sichergestellt sein, dass alle Daten immer auf dem neusten Stand sind. Dafür soll die Frequenz des Crawlers sich an dem Sitzungskalender des Bundestags [5] orientieren

2.2.2 Rahmenbedingungen

Organisatorische Rahmenbedingungen

Für die Planung des gesamten Projekts ist in einem zwei Wochen-Takt ein Plenum für das Brainstorming und eventuelle Abstimmungen der unterschiedlichen

Gruppen festgelegt worden. Im Team-Crawler (Gruppe 1) ist folgende Planung abgemacht worden:

- Ziel bis zum 13.11.2020: Testdaten für andere Gruppen, erster lauffähiger Prototyp
- Ziel bis zum 27.11.2020: Vorläufiger Release und Integration-Test mit anderen Gruppen (Kommunikationsmodell - Gruppe 2)
- Ziel bis zum 04.12.2020: Finale Version und Anfang der Dokumentation
- Ziel bis zum 25.02.2021: Fertigstellung der Dokumentation und Abgabe

Neben den organisatorischen Rahmenbedingungen sind die technischen ebenfalls zu betrachten. Die beim Crawl-Prozess auftretenden Probleme sind zu vermeiden oder müssen gelöst werden.

Technische Rahmenbedingungen: Probleme beim Crawlen

- **Sperrung durch Server-Administrator (Server-Regel):**
  Es soll vermieden werden, dass der ausführende Rechner aufgrund von zu häufigen Abfragen, welche nach einem bestimmten Muster ausgeführt werden, vom Server-Admin gesperrt wird. Eine Sperrung hätte zur Folge, dass Antworten des Servers immer mit einem HTTP-Statuscode 500 gesendet werden.
- **Ajax basierende Inhalte:** Die Open Data Webseite verwendet Ajax für die Bereitstellung der Daten und lädt diese im Hintergrund. Ajax ist ein Konzept zum asynchronen Datenaustausch zwischen Client und Server. Damit ist es möglich HTTP-Anfragen auszuführen während eine HTML-Seite angezeigt wird. Da somit die Webseite clientseitig gerendert wird, ist ein einfacher Download der Webseite nicht ausreichend. Des Weiteren nutzen die meisten Links, hinter denen Dateien stehen, Javascript. Ein Beispiel wären bestimmt Slides hinter welchen sich verborgene Regionen verbergen. Dies muss entsprechend berücksichtigt werden.
- **Bereitstellung:** Mongo-DB-Konfiguration und das zur Verfügung stellen der Daten für die Gruppe 2 des Kommunikationsmodells.

2.3 Lösung und Konzepte

2.3.1 Standard Aufbau eines Crawlers

Ein Crawler besteht in der Regel aus zwei wichtigen Teilen, einem Downloader und einem Scheduler. Der Downloader ist mit dem Webserver verbunden und lädt die Webseite(n) sowie alle erwünschten Dateien herunter. Der Scheduler ist, wie der Name schon sagt, für die Planung zuständig und erteilt den Tasks des Downloaders URLs, welches nach internen Priorisierung-Mechanismen festgelegt wird. (als Liste FIFO und als Graph mit DFS und BFS [6]–[7]).

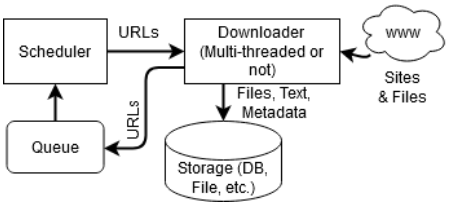

Abbildung 2.1: Crawler: Standard Aufbau

2.3.2 Crawler Mechanismus

In dem erstellten Projekt werden die Aufgaben des Schedulers vom Crawl-Manager übernommen, welcher sowohl für die Planung als auch für die Steuerung (Start, Pause, Stopp, Zustand) laufender Crawl-Aufgaben zuständig ist. Der Crawl-Manager erzeugt für jede URL ein Thread, einen sogenannten Page-Crawler, dessen Aufgabe darin besteht die Seite herunterzuladen, die Webseite mit dem Page-Analyser auf relevante Daten, wie URLs, Dateien und Links zu prüfen und bei Bedarf weitere parallele Sub-Threads zu erstellen. Diese Sub-Threads sind dann wiederum für den Download, das Parsen und die Speicherung der relevanten Informationen, wie Protokolle, Stammdaten und weiteren Metadaten zuständig. Dieser Mechanismus wird durch die Abb. 2.3 und die Abb. 2.2 veranschaulicht.

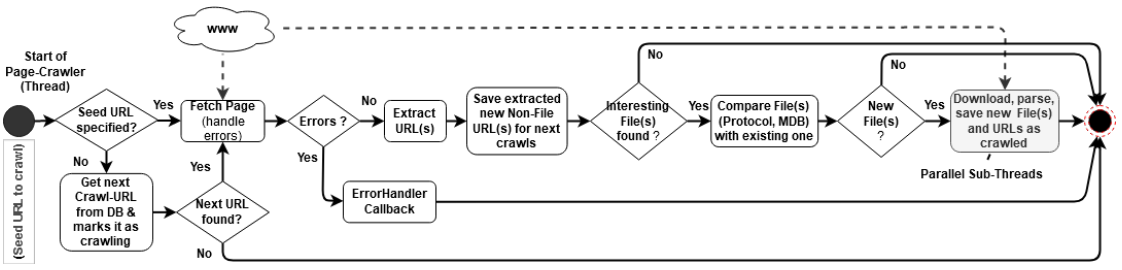

Abbildung 2.2: Page-Crawler für eine URL

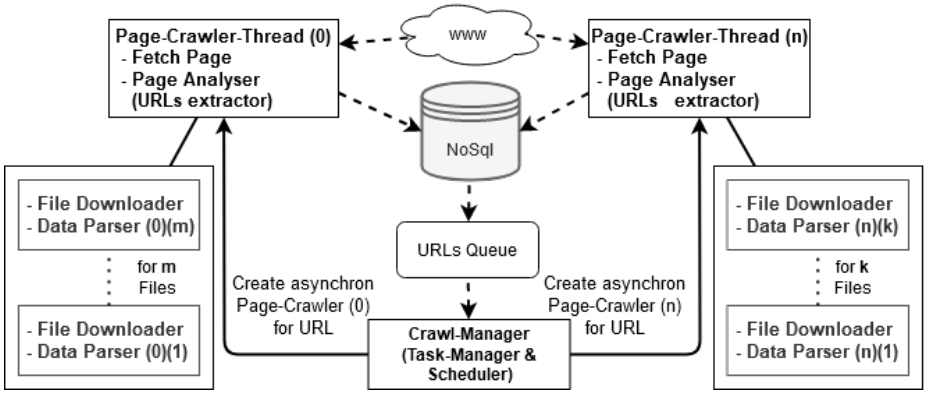

Abbildung 2.3: Funktionsprinzip des unseres Crawlers

Ein Crawl auf einer Seite, welche durch den Page-Crawler erfolgt, kann in einer durch den Scheduler bzw. Crawl-Manager regelmäßig gemacht werden, in dem dieser nach einer bestimmten Zeit einen neuen Page-Crawl-Thread startet. Die Frequenz der Generierung dieses Threads orientiert sich an dem Sitzungskalender des Bundestags [5]. Dieser Kalender sieht vor, dass Sitzungen an bestimmten Wochentagen jeweils zwei bis vier Wochen pro Monat stattfinden. Hiervon ausgenommen ist der August aufgrund der Ferienzeit. Darauf basierend ist die Entscheidung getroffen worden, die Frequenz auf einmal pro Tag von Montag bis Freitag festzulegen. Wobei das Stoppen des Crawlers jederzeit manuell möglich ist, wie zum Beispiel innerhalb der Ferien. So lässt sich eine Sperrung wegen zu häufiger Abfragen verhindern. Bei einigen Servern kann es jedoch vorkommen, dass durch bestimmte Serverregeln oder durch den Server-Admin ein sich wiederholendes Abfrage-Muster, zu einer Sperrung der IP-Adresse führen kann. Aus diesem Grund wird zusätzlich zur Frequenz ein Zufallsfaktor verwendet. Ein Page-Crawl-Thread wird von Montag bis Freitag jeweils um 23 Uhr durch den Crawl-Manager gestartet, allerdings wartet dieser Thread eine zufällige Zeit t (mit $10 \leq t \leq 7200$) bis der tatsächliche Crawl durchgeführt wird. Während und nach dem Crawl-Prozess werden Daten manipuliert, analysiert und gespeichert. Diese Vorgänge sowie die dafür verwendeten Datenstrukturen werden im nächsten Abschnitt näher erläutert.

2.3.3 Daten-Parser und Database-Modell

Für die Verwendung der vom Crawler heruntergeladenen Plenarprotokolle und Stammdaten müssen diese zunächst in verwendbare Daten umgewandelt werden. Dies übernimmt ein sogenannter Parser, welcher die brauchbaren Informationen aus den Dokumenten ausliest und diese in ein Modell integriert, welches in einer NoSQL-Datenbank gespeichert wird. Um zu verstehen wie der Parser funktioniert

und welche Daten relevant sind, muss zuallererst die Datenstruktur der Protokolle und Stammdaten analyisert werden, um auf dessen Basis das bereits genannte Modell zu entwickeln.
Im folgenden Abschnitt wird näher auf die Struktur der genannten Dateien eingegangen. In der Abbildung 2.4 ist die Datenstruktur der Stammdaten zu erkennen. In den Stammdaten stehen jegliche Informationen aller Abgeordneten, welche jemals teil des Bundestages waren. Im folgenden wird stichpunktartig auf die Einzelnen XML-Tags eingegangen:

- **MDB:** Das Stammdaten-Dokument besteht aus einer Liste dieses Tags, welche den Abgeordneten darstellt
- **ID:** Einzigartige Identifikationsnummer über welche der Abgeordnete in anderen Kontexten referenziert werden kann
- **NAMEN:** Stellt eine Liste von Namens-Objekten dar, welche mehr als einmal vorhanden sein können, falls sich Änderungen im Namen ergeben haben (z.B. durch eine Heirat)
- **NAME:** Enthält Informationen über den Vor-/Nachnamen, ggf. Präfixe, Titel, wann die Änderung des Namens eingetreten ist, falls es eine gab, etc.
- **WAHLPERIODEN:** Stellt eine Liste von Wahlperioden-Objekten dar, welche angeben in welchen Wahlperioden der Abgeordnete gewählt worden ist
- **WAHLPERIODE:** Enthält Informationen über die Wahlperiode in welcher der Abgeordnete tätig war. Dazu gehören Informationen wie die jeweilige Wahlperiode, Anfang und Ende der Periode, Wahlkreisland und -nummer, als auch die Mandatsart und Daten zur Institution.
- **INSTITUTIONEN:** Stellt eine Liste von Institutions-Objekten dar, welche mehrmals vorhanden sein können, falls der Abgeordnete seine Institution innerhalb der Wahlperiode geändert haben sollte
- **INSTITUTION:** Enthält Daten zur Institution, welche der Abgeordnete zur jeweiligen Wahlperiode angehörte. Dazu gehören Informationen wie Name der Institution (meist der Fraktionsname), Eintritts- und Austrittsdatum, etc.
- **BIOGRAFISCHE DATEN:** Enthält viele Daten über den Abgeordneten und dessen Leben, wie Geburtsland, -ort, datum, Sterbedatum, Religion, Beruf, Familienstand, Geschlecht, etc.

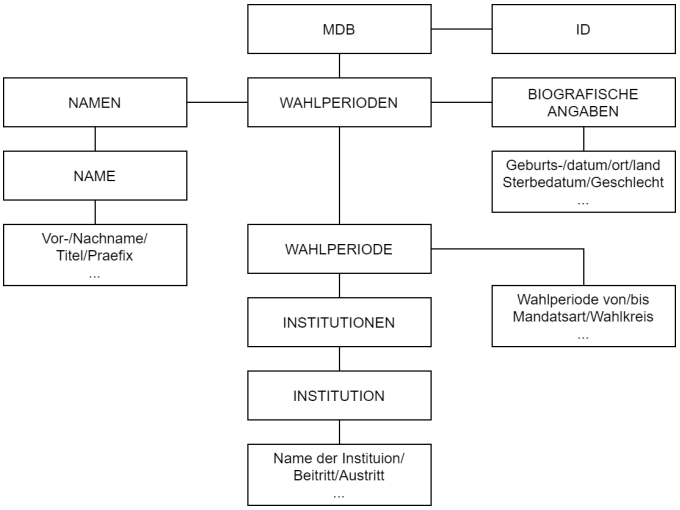

Abbildung 2.4: Stammdaten Struktur

Aus den vielen Daten der Stammdaten über die Abgeordneten lassen sich unter Umständen einige Erkenntnisse in den späteren Sentimentanalysen verwerten. Für weitere Informationen zur Struktur der Stammdaten kann die DTD-Datei auf der Bundestagsseite eingesehen werden.

Die Plenarprotokolle der 19. Legislaturperiode sind ebenfalls im XML Format verfasst, bei welchen die Datenstruktur übersichtlich dargestellt werden kann. Im folgenden wird auf die Struktur des Plenarprotokolls eingegangen. Bei der in Abbildung 2.5 aufgeführten Struktur handelt es sich um eine grob dargestellte Variante, welche nur die für den Parser und somit für das Projekt interessanten Daten aufzeigt. Somit wird im folgenden Abschnitt nicht auf die XML-Tags 'vorspann' und 'anlagen' eingegangen.

- **dbtplenarprotokoll:** Enthält alle Informationen über die Bundestagssitzung
- **rednerliste:** Enthält alle Redner, welche in der Sitzung, zu welcher das Protokoll gehört, eine Rede gehalten haben.
- **redner <id>:** Beschreibt den Redner und enthält grobe Informationen über diesen. Zeigt ebenfalls die zugehörige einzigartige ID des Redners, welche in den Stammdaten aufgeführt ist
- **redner/name:** Enthält Informationen wie Vor-, Nachname, Titel und die Fraktion des Abgeordneten

- **sitzungsverlauf:** Enthält alle protokollierten Informationen der Sitzung. Dazu gehören alle angesprochenen Punkte, Reden und Kommentare der teilnehmenden Abgeordneten

- **sitzungsbeginn:** Beginn der Sitzung, welche meist von dem Bundespräsidenten eingeleitet wird.

- **tagesordnungspunkt:** Informationen über den jeweiligen Tagesordnungspunkt. Unter diesem werden jeweils mehrere Reden von Abgeordneten gehalten auf welche Kommentare und Reaktionen folgen können

- **rede <id>:** Die jeweilige Rede eines Abgeordneten, dessen Inhalt und Reaktionen. Des Weiteren besitzt jede Rede eine chronologische ID, welche nur innerhalb des Protokolls einzigartig ist

- **p:** Dieser Tag stellt einen Paragraphen dar, wobei dessen Inhalt eine Aufnahme der gesagten Worte des jeweiligen Redners darstellen

- **p <klasse='redner'>:** Dieser Tag mit der Klasse 'redner' ist der erste Paragraph einer Rede und stellt den Redner dar, welcher die Rede führt. Der Redner wird mit groben Informationen und seiner Identifikationsnummer näher beschrieben

- **kommentar:** Kommentar eines Abgeordneten oder einer Fraktion, welches meist aufgrund einer Rede hervorgerufen wird. Dies können Tätigkeiten, wie Beifall oder Zurufe sein.

- **name:** Der Inhalt dieses Tags ist ein einfacher Name desjenigen, welcher innerhalb oder auch außerhalb einer Rede anfängt etwas zu sagen. Anders als ein aufgeführter Redner, führt diese Person meist nur einen kleinen Redeanteil, welcher nicht als eigenständige Rede aufgefasst wird. Ein Beispiel dafür wären die Worte des Bundespräsidenten zur Einleitung der Tagesordnungspunkte.

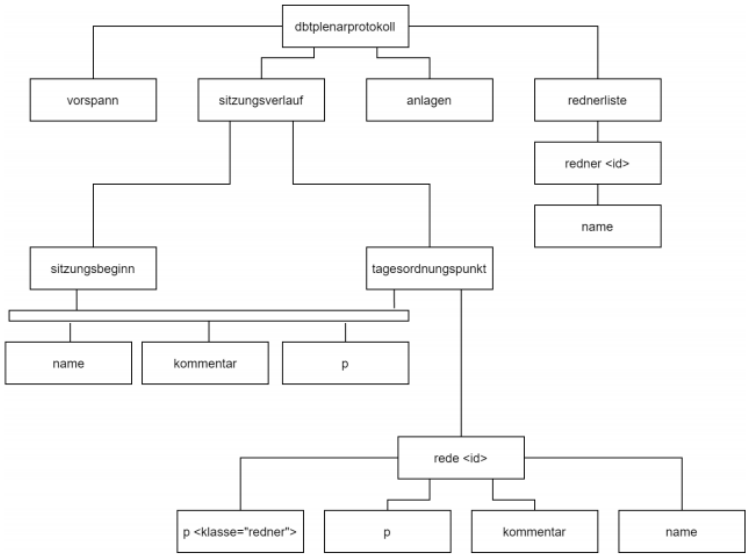

Abbildung 2.5: Plenarprotokoll Struktur

Aus der Analyse der beiden Strukturen wurde das Datenbankmodell, welches in Abbildung 2.6 abgebildet ist, erstellt.

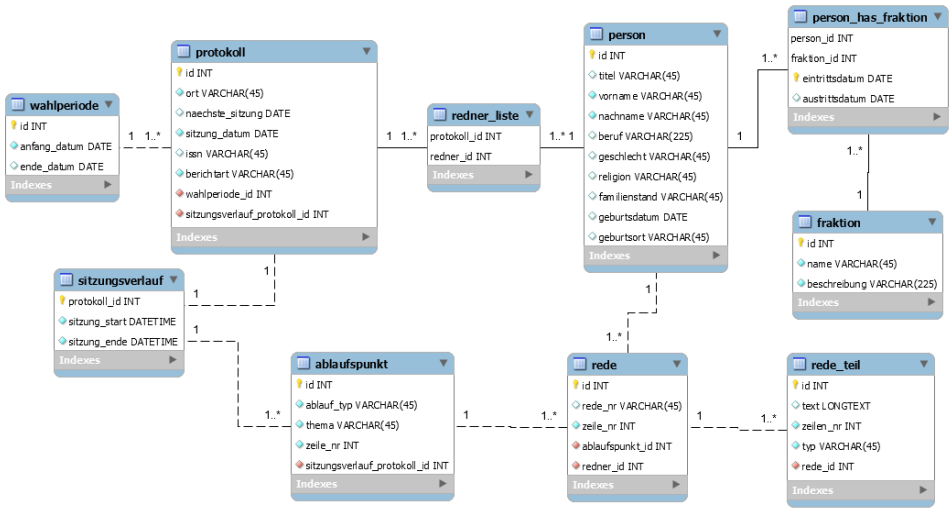

Abbildung 2.6: Datenbankmodell

In diesem Abschnitt wird nur kurz auf das vorliegende Datenbankmodell eingegangen, da die meisten Tabellen und Spalten selbsterklärend sind. Die Spalte 'id' beschreibt in der Tabelle 'wahlperiode' die jeweilige Legislaturperiode, so wie in der Tabelle 'protokoll' die Sitzungsnummer. Die gesammelten Informationen über

die Abgeordneten aus den Stammdaten werden in der Tabelle 'person' gespeichert. Des Weiteren werden deren Fraktionszugehörigkeit, als auch gegebenenfalls Wechsel dieser mit aufgenommen. Die gespeicherten Abgeordneten können nun im späteren Verlauf durch die in der Rednerliste der Protokolle abgespeicherten Identifikationsnummern der Redner referenziert werden. Die Tagesordnungspunkte und der Sitzungsbeginn eines Protokolls werden aufgrund ihrer starken Ähnlichkeit zueinander zu einer Tabelle namens 'ablaufspunkt' zusammengefügt. Lediglich über die Spalte 'ablauf_typ' ist zu erkennen, ob es sich um einen Tagesordnungspunkt oder ein Sitzungsbeginn handelt. Ebenfalls wurden Paragraphen, welche Redeanteile beinhalten, und Kommentare in der Tabelle 'rede_teil' zusammengefügt, welche ebenfalls über den Eintrag 'typ' unterschieden werden können. Die Spalte 'paragrafKlasse' in der eben genannten Tabelle gib die 'klasse' des XML-Tags 'p' wieder. Ein Beispiel dazu wäre die in der Protokoll Struktur genannte Darstellung eines Redners über den XML-Tag 'p<klasse='redner'> innerhalb des Protokolls. Die Klasse des Paragraphen beschreibt diesen teils näher. Diese Eigenschaft wurde der Tabelle hinzugefügt aufgrund einer potenziell besseren Verarbeitung der Daten zum Erstellen des Kommunikationsmodells. Des Weiteren werden jeweils die Zeilennummern mitgespeichert, um eine chronologische Reihenfolge der Kommunikationsstruktur zu erhalten.
Wie in der Einführung erwähnt müssen die Daten aus den Protokollen mithilfe eines Parsers in ein Modell umgewandelt werden, welches anschließend in der Datenbank gespeichert werden kann. Der Parser dieses Projektes wurde in Java mithilfe des 'Java DOM Parsers' aus der 'JavaX' Bibliothek umgesetzt. Mittels des Parsers wird das XML-Dokument in ein 'Document Object Model' (DOM) umgewandelt. Über die jeweiligen XML-Tags kann dann auf den Inhalt dieser zugegriffen werden. Des Weiteren werden alle XML-Dokumente vor dem Parsen formatiert. Dies ist eine Erweiterung, welche nach einem Problem hinzugefügt worden ist. Aufgrund eines Protokolls, welches vom Ersteller nicht richtig formatiert worden war, kam es zu Problemen der Zeilennummern, da mehrere Tags und somit Inhalte, in einer einzigen Zeile geschrieben worden waren, welches dazu führte, dass keine chronologische Reihenfolge der Kommunikationsstruktur mehr nachvollziehbar war. Abschließend ist zu erwähnen, dass der Parser nur für die Protokolle und XML-Struktur der 19.Legislaturperiode ausgelegt ist. Wie jedoch bereits in diesem Dokument erwähnt, wird parallel zu diesem Projekt an einer Lösung gearbeitet, welches die Protokolle der 18.Legislaturperiode mithilfe einer KI in die selbe XML-Struktur formatiert. Nach einem Erfolg des eben genannten Projektes wird es ebenfalls möglich sein die 18. Periode zu parsen und diese für weitere Analysen zu verwenden.

2.3.4 Gesamter Aufbau der Lösung

Die gesamte Lösung besteht aus vier Komponenten und einer Datenbank, wie in der Abb. 2.7 dargestellt.

- **Crawl-Manager**: Taskmanager und Scheduler
- **Crawl-Utilities**: Liefert die für den Crawl-Prozess nötigen Funktionalitäten (Page-Fetcher, Page-Analyser, Downloader, Data-Parser)
- **DB-Manager**: Verwaltet den Zugriff auf die Datenbank
- **Rest-ServiceProvider**: Rest-API für die Steuerung des Crawl-Managers und des eingeschränkten Zugriffs auf die DB-Daten
- **Datenbank**: NoSql (MongoDB) Datenbank zur Sicherung der Daten

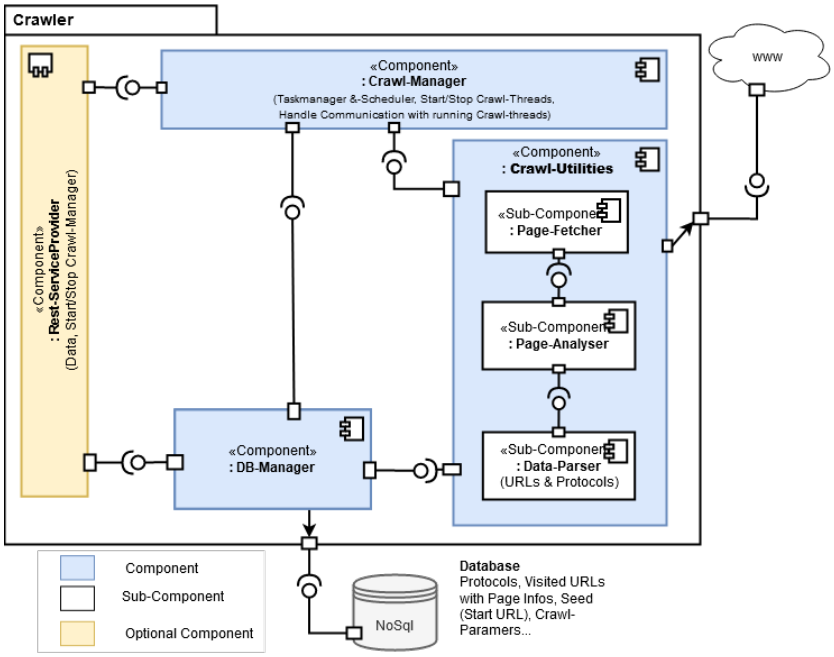

Abbildung 2.7: Crawler: Komponentendiagramm

Am Ende eines Crawl-Prozesses, bei welchem neue Protokolle oder Stammdaten geladen worden sind, wird die Gruppe des Kommunikationsmodells über eine REST-API [8] benachrichtigt. In dieser Benachrichtigung befindet sich eine Liste aller neuen Protokolle sowie Stammdaten, welche als JSON versendet werden. Die Gruppe-Kommunikationsmodell kann im Anschluss asynchron die entsprechenden Dateien aus der bereitgestellten Datenbank laden.

2.4 Implementierung und Bereitstellung

2.4.1 Implementierung des Crawlers

Für die Umsetzung des Crawlers wurde die Entscheidung getroffen eine Webanwendung zu implementieren. Für dieses Vorhaben wurde Spring Boot [9] gewählt, welches ein auf der Programmiersprache Java basierendes Framework für die Entwicklung von Webanwendungen ist. Mithilfe des Frameworks lassen sich Konfigurationen für REST-APIs, Datenbanktreiber, etc. über den Spring-Initializer vereinfachen. Außerdem verfügt Spring Boot über einen eigenen Task-Scheduler der für den Crawl-Manager verwendet werden kann. Die finale Lösung beinhaltet vier Packages:
crawler mit den Funktionalitäten für den Crawl-Prozess und das Parsen, **models** und **repositories** für den DB-Manager. Des Weiteren das Package **web** mit den Controllern für die Rest-Schnittstelle und den Services für den Crawl-Manager. Das gesamte Projekt-Verzeichnis inklusive des Codes ist auf dem Github-Repository *Sentiments-of-Bundestag/Crawler* [10] zu finden. Für den Download bestimmter Informationen ist es nötig auf Javascript basierende Inhalte auszuführen. Um dies durchzuführen wird die
HtmlUnit-Bibliothek [11] genutzt. (siehe 2.2.2).

2.4.2 Bereitstellung der Lösung

Der Crawler wurde auf dem Virtual-Server der Gruppe 1 (141.45.146.161) an der HTW als Ubuntu-Service bereitgestellt. Dieser ist aus Sicherheitsgründen nur im HTW-Netzwerk sichtbar. Der Crawler-Service liefert eine REST-Schnittstelle, die jedoch aus Sicherheitsgründen nur auf dem infosys1 Rechner zugänglich ist. Die Konfiguration des Crawler-Services, die unter */etc/systemd/system/crawler.service* angelegt wird, sieht wie folgt aus (Start des Crawlers: systemctl enable crawler):

```
1 [Unit]
2 Description=Crawler App
3 User=local
4 [Service]
5 ExecStart=/usr/bin/java -jar ../Crawler-1.0-SNAPSHOT.jar
6 Restart=always
7 SyslogIdentifier=crawler
8 [Install]
9 WantedBy=multi-user.target
```

Da für den Crawler eine MongoDB benötigt wird, soll diese auch bei der Bereitstellung konfiguriert werden. Dabei muss sichergestellt werden, dass der Zugriff auf die Datenbank nur mit den entsprechenden Zugangsdaten erfolgt. Somit ist

zu beachten, dass nach dem Hinzufügen eines neuen Admin-Users die Security-Konfiguration von mongod unter *sudo nano /etc/mongod.conf* auf *authorization: enabled* umgestellt werden. Da die MongoDB von der Gruppe 2 (Kommunikationsmodell) verwendet wird, sollen dafür entsprechend die Firewall-Regeln angepasst werden. Der nachfolgende Auszug aus der Konfiguration-Datei (/root/Firewall.sh) zeigt wie dies beispielhaft erfolgen sollte.

```
# Erlaube zugang im 145.45.x.x port 27017 MongoDB
iptables -A INPUT -s 141.45.0.0/16 -p tcp -m tcp --dport 27017 -j
    ACCEPT
```

Der bereitgestellte Crawler basiert auf Java 11 und ist somit eine plattformunabhängige Lösung, die auf allen Betriebssystemen angeboten werden kann. Die Steuerung des Crawlers erfolgt ausschließlich über die REST-Schnittstelle mit folgenden Abfragen (hier mit curl in der Shell-Konsole):

```
# opendata: https://www.bundestag.de/services/opendata
# Start a default crawl to opendata
$ curl -X POST http://localhost:8080/task/default
# Start the default cron process to opendata: Mon - Fri, 23
$ curl -X POST http://localhost:8080/task/cron
# List all running tasks
$ curl -X GET http://localhost:8080/tasks
# Cancel planed task
$ curl -X POST http://localhost:8080/task/cancel/{task_id}
# Cancel all planed tasks
$ curl -X POST http://localhost:8080/tasks/cancel
```

2.5 Zusammenfassung und Ausblick

Trotz einiger Schwierigkeiten im Zusammenhang mit der Struktur der OpenData Webseite [2] und im Abschnitt 2.2.1 erwähnt konnte mit Hilfe verschiedener Technologien und Frameworks (Spring Boot, Taskscheduler und HtmlUnit) eine leistungsfähigen, Multitasking-fähige Crawler-Lösung (Software as a Service (SaaS)) umgesetzt werden. Der Crawler auf dem infosys1-Server (infosys1.f4.htw-berlin.de) läuft seit dem 27.11.2020, bis auf kleine Unterbrechungen aufgrund von Aktualisierungen, problemlos und konnte zum Zeitpunkt der Verfassung dieser Dokumentation schon über 210 Protokolle herunterladen, parsen und in der MongoDB speichern. Des Weiteren wurden Stammdaten von 4.086 Abgeordneten gesammelt und mehr als 540 URLs durchsucht. Der Datenaustausch mit der Gruppe des Kommunikationsmodells war erfolgreich, da diese ohne Probleme auf die bereitgestellte MongoDB zugreifen konnten und die Benachrichtigungen ebenfalls fehlerfrei empfangen wurden.
Aus den vom Crawler gesammelten Daten wird nun durch Gruppe 2 ein Kommunikationsmodell abgeleitet, welches von anderen Gruppen für weitere Verarbeitungen und Analysen verwendet werden kann. Der Aufbau dieses Kommunikationsmodells wird nun von Gruppe 2 im nächsten Kapitel aufgegriffen.

Kapitel 3

Kommunikationsmodell

Gruppenmitglieder: Max Lüdemann, Oskar Sailer, Ralph Schlett, Youri Seichter

3.1 Einleitung

Im Rahmen des Projekts Sentiments of Bundestag (SoB) sollten Bundestagsprotokolle analysiert werden, um den „Ton“ des Bundestags zu bewerten. Das in diesem Kapitel beschriebene Teilprojekt, das von Gruppe 2 bearbeitet wurde, hatte die Wahl und Anwendung eines passenden Kommunikationsmodells zum Ziel. Zur Erfüllung dieser Zielstellung wurde die „Communication Model Extractor (CME)“-Software erstellt. Diese ist das letzte Pipeline-Element des Gesamtprojektes, das sich mit den Rohdaten der Bundesregierung beschäftigt. Die in diesem Teilprojekt generierten Daten stellen die Basis aller weiteren Verarbeitungsschritte dar.

3.1.1 Zielstellung

Die CME-Software soll aus den *Open Data* [2] Extensible Markup Language (XML)-Dateien der 19. Wahlperiode des Bundestages bzw. der leicht abgewandelten Variante dieser Daten, welche von Gruppe 1 in Form von „JavaScript Object Notation (JSON)“-Dateien zur Verfügung gestellt werden, Interaktionen zwischen Fraktionen und oder Abgeordneten extrahieren.

Für diese Extraktion werden die Rohdaten analysiert und darauf aufbauend ein passendes Kommunikationsmodell gewählt. Um über neue Protokolle von Gruppe 1 informiert werden zu können, soll dafür eine Schnittstelle geschaffen werden. Ebenfalls werden zur Weitergabe der Ergebnisse an die folgende Gruppe ein Datenmodell, welches zur Speicherung der Ergebnisse dient, eine Schnittstelle zur Weitergabe der Daten und eine weitere Schnittstelle, über welche Gruppe 3 bei neuen Daten benachrichtigt werden kann, entwickelt.

3.1.2 Anforderungsdefinition

Auf Grundlage der im vorherigen Abschnitt beschriebenen Zielsetzung sowie der Diskussionen in den Plenarsitzungen während des Semesters, erfolgt in diesem Kapitel eine Zusammenfassung von funktionale Anforderungen, welche an das zu entwickelnde System gestellt werden. Diese werden in Muss-, Soll- und Kann-Kriterien unterteilt.

Tabelle 3.1: Funktionale Anforderungen

Muss-Kriterien	**FR01**	Ein passendes Kommunikationsmodell und Datenmodell muss gewählt werden
	FR02	Das Ausgabeformat von Gruppe 1 muss eingelesen werden können
	FR03	Interaktionen basierend auf den Kommentaren zu Redebeiträgen müssen extrahiert werden
	FR04	Extrahierte Interaktionen müssen für den Zugriff späterer Gruppen persistiert werden
	FR05	Spätere Gruppen müssen auf die persistierten Nachrichten zugreifen können
	FR06	Gruppe 1 muss die Möglichkeit haben, uns über die Verfügbarkeit neuer Daten zu benachrichtigen
	FR07	Gruppe 3 muss von uns benachrichtigt werden, wenn neue Daten zur Verfügung stehen
	FR08	Parteien und Abgeordnete müssen über Sitzungen hinweg eindeutig identifizierbar sein
	FR09	Daten von Gruppe 1 müssen aus deren MongoDB ausgelesen werden können
Soll-Kriterien	**FR10**	Das Open Data Ausgabeformat des Bundestags soll eingelesen werden können
Kann-Kriterien	**FR11**	Interaktionen innerhalb der Redebeiträge können extrahiert werden

3.1.3 Wahl des Kommunikationsmodells

FR01 erfordert die Auswahl eines passenden Kommunikations- und Datenmodells. Aus diesem Grund wurden zu Beginn des Semesters die bereits vorliegenden Plenarprotokolle im „*Open Data*“-Format der Bundesregierung untersucht.

Bei dieser Untersuchung zeigte sich, dass die für uns interessanten Daten so angeordnet sind, dass jede Sitzung in Tagesordnungspunkte unterteilt ist, welche wiederum aus einer Vielzahl von Reden besteht. Eine solche Rede enthält dabei Informationen über den Redner, die in einzelne Absätze zerteilte Rede und op-

tionale Kommentare anderer Parlamentarier oder Fraktionen zu diesen Absätzen. So sind in diesen Strukturen Interaktionen erkennbar, z. B. in den Kommentaren an den Redner gerichtete Interaktionen oder auch in den Absätzen Interaktionen zwischen dem Redner und anderen Parlamentariern.

Basierend auf diesen Erkenntnissen und in Absprache mit Gruppe 3, 4 und 5, welche für die Sentimentanalyse sowie die Auswertungen zwischen Fraktionen und Abgeordneten verantwortlich sind, wurde ein Sender-Empfänger-Modell für die Repräsentation der Interaktionen gewählt. In diesem kann der Sender und Empfänger jeweils entweder ein Bundestagsabgeordneter oder eine Fraktion sein. Auf Basis des Inhalts der Nachricht, die vom Sender zum Empfänger geschickt wird, wird später von Gruppe 3 das Sentiment ermittelt.

3.2 Umsetzung

Auf Grundlage der definierten Anforderungen wurde im nächsten Schritt ein Konzept für deren Umsetzung entwickelt. Dieses sowie die Umsetzung selbst wird in den folgenden Abschnitten näher erläutert.

3.2.1 Communication Model Extractor (CME)

Für die Entwicklung der CME-Software wurde die Programmiersprache Python gewählt, da alle Teammitglieder bereits Kenntnisse in dieser mitbrachten und um den folgenden Gruppen möglichst schnell erste Ergebnisse liefern zu können, mit denen diese arbeiten können. So bietet Python eine Vielzahl an guten und einfach zu nutzenden Bibliotheken, welche die Entwicklung einer solchen prototypischen Applikation beschleunigen können. Z. B. ist es innerhalb weniger Zeilen möglich, einen funktionsfähigen Representational State Transfer (REST)-Endpoint zur Verfügung zu stellen.

Aus den funktionalen Anforderungen FR05, FR06 und FR07 ergibt sich, dass eine Kommunikation zu den Gruppen 1 und 3 nötig ist. Aufgrund hoher Kompatibilität wurde sich für eine REST-Schnittstelle entschieden. Diese API wurde in dem Python-Modul „api.py“ mithilfe der FastAPI-Bibliothek implementiert. Sie beantwortet Anfragen zu Protokollen (Sessions), Fraktionen (Factions) und Mitglieder des Deutschen Bundestages (MDBs). Außerdem wird eine SWAGGER-Dokumentation bereitgestellt.

Zur Steuerung der Anwendung wird ein Command Line Interface (CLI), welches im „cli.py“-Modul umgesetzt wurde, in Form der CME-Applikation zur Verfügung gestellt. Dieses ist in verschiedene Unterkommandos aufgeteilt. Mithilfe dieser ist es z.B. möglich Protokolle manuell aus dem lokalen Speicher einzulesen (XML- oder JSON-Format) oder den Server zu starten. Mit „init“ werden erstma-

lig Bundestagsabgeordnete eingelesen. „dump“ ermöglicht die Interaktion mit der Datenbank, um einzelne Dokumente daraus zu extrahieren.

Um die Anwendungslogik von der Business-Logik zu trennen, wird der „Controller“ eingeführt. Es ist das zentrale Modul, welches die Prozesse koordiniert.

Für die generische Verarbeitung der Rohdaten in den verschiedenen Datenformaten von Gruppe 1 (FR02) und der Open Data Protokolle des Bundestags (FR10) wurde das „Data“-Modul implementiert. Dieses wandelt die Rohdaten unter anderem in sogenannte InteractionCandidates um, welche dann mithilfe des „Extraction“- Moduls verarbeitet und ausgewertet werden (FR03 und FR11). So extrahiert dieses aus den InteractionCandidates die stattgefundenen Interaktionen basierend auf dem Sender-Empfänger-Modell und legt diese in einem sogenannten Transcript ab. Dieses Transcript wird anschließend mithilfe des „Database“-Moduls, welches für jeglichen Datenbankzugriff zuständig ist, in der Datenbank abgelegt.

Zuletzt enthält „Domain“ Datenmodelle für verschiedene Objekte, wie z. B. MDB, Faction & Interaction. Es wird von so gut wie jedem Modul benutzt.

Die soeben beschriebenen Module und der damit verbundene Kontrollfluss zwischen diesen ist in Abbildung 3.1 dargestellt.

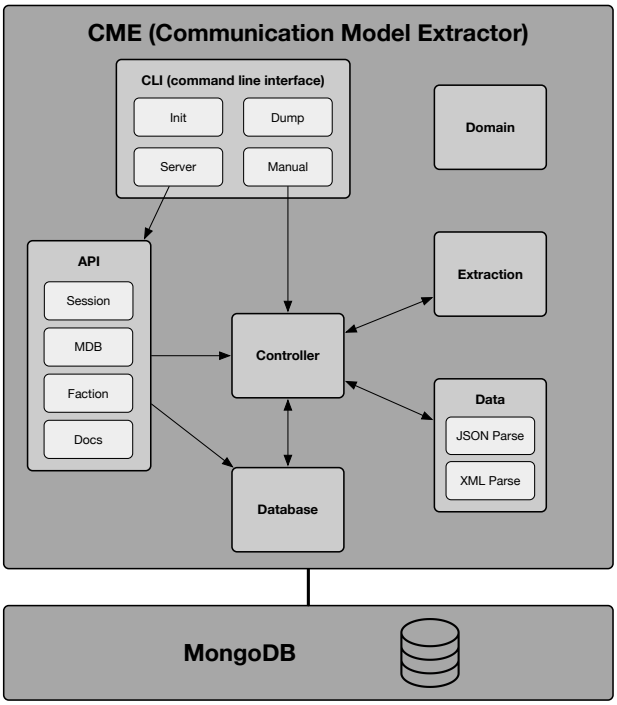

Abbildung 3.1: Applikations-Architektur

3.2.2 Eingabeformate

Wie in den Anforderungen FR02 und FR10 definiert, muss als Eingabeformat einerseits das JSON-Format der Gruppe 1 und andererseits soll auch das Open Data XML-Format des Bundestags unterstützt werden. So wurden spezielle Parser für die jeweiligen Formate entwickelt, deren Ausgabe gleich formatiert ist und somit für die nächste Instanz eine einheitliche Schnittstelle bildet. Dieses Ausgabeformat enthält dabei neben Metadaten zur Sitzung selbst, wie z. B. den Sitzungsstartzeitpunkt oder die Sitzungsnummer, sogenannte InteractionCandidates. Diese gruppieren, wie der Name bereits nahelegt, Daten, welche eventuelle, für die Auswertung relevante, Interaktionen enthalten. So gruppiert ein InteractionCandidate den Redner (Speaker), einen Absatz von dessen Rede (Paragraph) und den eventuell darauf folgenden Kommentarblock (Comment).

3.2.3 Methodik der Kommunikations-Analyse

Die zu analysierenden Daten (InteractionCandidates) bieten zwei verschiedene Kommunikationstypen als Datengrundlage für die Erkennung von Interaktion:

1. **Kommentare (FR03)**: werden von den Redeteilen gesondert dargestellt und mit zusätzlicher Information bezüglich der kommunikativen Einordnung des Beitrags versehen.

2. **Redeteile (FR10)**: werden in den Protokollen als Paragraphen oder sinngemäß als Absätze aufgeführt.

Aufgrund der Unterschiedlichkeit mussten beide Typen gesondert analysiert werden.

Kommentare

Kommentare werden nach rigiden strukturellen Regeln in den Protokollen der Bundestagssitzungen erfasst und liefern unter anderem Informationen zum Sender des Kommentars. Dieser kann eine oder mehrere Fraktionen oder eine oder mehrere an der Sitzung teilnehmende Personen sein. Kommentare lassen sich in die Folgenden drei Kategorien unterteilen:

- **Publikumsresonanzen**: Z. B. Beifall oder Entrüstung des Senders, bei der eine genaue Zitierung durch die Stenographen nicht möglich ist. Hier folgen Sender durch Präpositionen abgetrennt auf den Kommentarinhalt. Bsp.: „Beifall bei der CDU“; „Zurufe von der Linken und Grünen“

- **Zitate**: Hier wird der Wortlaut eines Kommentars wiedergegeben. Bei diesen Kommentaren wird die kommentierende Person dem Wortlaut vorangestellt und durch einen Doppelpunkt abgetrennt. Bsp.: „Dr. Klaus Hermann: Ganz toll gemacht“; „Dr. Alexander Gauland: Schwachsinn“
- **Beobachtungen der Protokollanten**: Spezielle Situationen in denen ein oder mehrere Sitzungsteilnehmer nennenswerte Handlungen vornehmen. Auch hier wird in der Regel der Sender der Nachricht vorangestellt, wenngleich keine Trennung durch einen Doppelpunkt erfolgt. Bsp.: „Manfred von Kuchenhausen verlässt die Sitzung“

Je nach erkannter Kommentarform kommen also verschiedene strukturelle Analyse-Vorgänge für die Feststellung von Sendern zum Einsatz.

Kommentare mit abweichender Struktur Es gibt weitere Eigenschaften, die die Extraktion der Interaktionen erschweren. So können pro Kommentar-Abschnitt mehrere Strukturen mit unterschiedlichen Inhalten auftreten, oder es werden insbesondere bei Publikumsresonanzen mehrere Absender aufgeführt (etwa „Beifall von SPD und die Linke“), wodurch zusätzlich eine korrekte Auftrennung der Sender auf Basis von Konjunktionen bzw. Interpunktion notwendig wird. Außerdem gibt es gelegentlich sekundäre Kommentare, bei denen auf vorherige Kommentare reagiert wird. Die einzigen zuverlässigen Formatierungen, die auf solche Verhältnisse hinweisen, sind die expliziten grammatikalischen Strukturen nach der Namensnennung: „an [neuer Empfänger] gerichtet“ bzw. „zur [Ziel-Fraktion] gewandt“ wirken innerhalb von Kommentaren als klares Merkmal für einen vom Redner abweichenden Empfänger. Das Schlüsselwort „Gegenruf“ wird darüber hinaus häufig verwendet, um auf einen Dialog zwischen Kommentierenden hinzuweisen - da in diesen Fällen allerdings der Kommentarinhalt oft mindestens teilweise weiterhin auf den Redner abzielt, wurde bei solchen Kommentaren ebenfalls der Redner als Empfänger verwendet. Diese Problematik könnte in Fortsetzungen der Arbeit eventuell präziser behandelt werden, da in längeren Kommentar-Dialogstrukturen recht häufig auf die eindeutige Empfängerbezeichnung verzichtet wird. Dadurch werden in unserer Methode viele für menschliche Leser offensichtliche Interaktionen nicht als solche registriert. Nicht zuletzt werden die genannten syntaktischen Strukturen gelegentlich durch Rechtschreib- oder Zeichensetzungsfehler durchbrochen, wobei in diesen Fällen die gesamte Interaktion verloren geht.

Redeteile

Redeteile besitzen weniger Struktur als Kommentare, da sie nicht den Regeln der Stenographen unterliegen, sondern des individuellen Redners. So muss der Empfänger einer Aussage erst ermittelt werden. Redeteile wurden daher auch nur als

Interaktionen gewertet, sobald Fraktionen oder Mitglieder des Bundestags als mögliche Interaktionsempfänger im Fließtext erkannt wurden.

Die Erkennung von MDBs erfolgt über die Feststellung bekannter, eindeutiger Nachnamen, denen eine formale Anrede (Herr/Hr., Frau/Fr., Kollege, Doktor) vorangeht. Parteien wurden anhand von ihren Namen bzw. deren Abkürzungen, beziehungsweise durch gängige aber eindeutige Kurzformen identifiziert. Die festgestellte Person oder Partei wurde anschließend als Empfänger der Nachricht behandelt, während die sprechende Person, also der durch den Sitzungspräsidenten zuletzt eingeleitete Redner, die Rolle des Absenders einnahm.

Die Empfängerermittlung bei Redeanteilen verblieb in der entstandenen Implementierung in einem recht rudimentären Zustand, da viele der inhärenten Probleme des Ansatzes nicht behandelt werden konnten. So hat etwa die Unterscheidung zwischen mehreren MDBs mit gleichen Nachnamen oder die kolloquiale Verwendung von Farben („Rot“ für SPD oder die Linke) oder Begriffen („Liberale“ stellvertretend für die FDP) als Substitut für Fraktionen zur Generierung vieler falscher Interaktionen geführt. So kann z. B. „Rot“ einerseits in Kontexten wie „roter Gentechnik“ und andererseits zur Nennung der SPD genutzt werden. Aus diesem Grund wurde die Funktionalität wieder entfernt. Daher werden nur solche Interaktionen gewertet, welche sich an eindeutig identifizierbare MDBs oder Fraktionen richten. Nachnamen wie „Müller“, die mehrere MDBs bezeichnen können, aber auch in anderen Sachverhalten eingesetzte Begriffe, wie „Union“, wurden somit aus der Schlüsselwort-Liste entfernt, mit der die Redeabschnitte abgetastet wurden. Es sollte allerdings nicht außer Acht gelassen werden, dass Interaktionen in Redeteilen nach unseren Messungen einen recht geringen Anteil der Interaktionen im Bundestag ausmachen. Die Steigerung der Anzahl erkannter Interaktionen durch unsere Methode der Redebeitrag-Analyse betrug für die bisherigen Protokolle der 19. Wahlperiode lediglich 8% gegenüber der alleinigen Betrachtung der Kommentare, was dafür spricht, dass auch nach Aufnahme von weiteren Fraktionsbegriffen bzw. Nachnamen in die Betrachtung nur relativ wenige Interaktionen hinzukommen würden.

3.2.4 Ermittlung von teilnehmenden Personen

Personen müssen über Sitzungen hinweg eindeutig identifizierbar sein, damit nachfolgende Gruppen Informationen zu Parteizugehörigkeit und Namen korrekt zuweisen können (FR08). Personen werden in der „mdb“-Collection der MongoDB gesammelt. Initialisiert werden kann diese Liste mithilfe der Stammdaten der Bundestagsabgeordneten des Bundestags. Diese, als XML zur Verfügung gestellte, Liste enthält alle relevanten Informationen, die bei der Identifizierung helfen (Namen und Namensänderungen, Parteizugehörigkeit mit von-bis-Datumsangabe, Titel etc.).

Da die Stammdaten unvollständig sind, auch weil Gastredner Reden halten oder Bundestagsabgeordnete in den Stammdaten fehlen, müssen weitere Redner während der Analyse hinzugefügt werden. Gerade in den Kommentaren können meist Vor- und Nachnamen extrahiert werden, um ein Abgleich mit den MDB-Daten vorzunehmen. Leider werden Namen oft inkonsistent geschrieben (z. B. wird der zweite Vorname manchmal ausgeschrieben, abgekürzt oder weggelassen) oder die Stenographen vertippen sich. Auch wird das Format der Kommentarsektion nicht immer eingehalten.

Weil fehlende Redner und Rechtschreibfehler nicht unterschieden werden können, gelangen doppelte Einträge in die Datenbank. Es wurden Mechanismen eingebaut, um solche Fehler zu erkennen. Diese können jedoch nicht ganz ausgeschlossen werden.

3.2.5 Infrastruktur-Setup mit Docker

Um ein einfaches Deployment zu ermöglichen, wurde ein Container-basierter Deploymentansatz gewählt. Dieser nutzt sowohl einen Container für die im Rahmen des Teilprojektes erstellte Software, als auch für die MongoDB, welche für die Persistierung der Daten genutzt wird.

Die beiden Container werden mithilfe von docker-compose verwaltet und konfiguriert. Dabei gibt es neben der docker-compose-Konfiguration für das Deployment auch eine zweite für die lokale Entwicklung. Diese enthält primär die MongoDB, da während der Entwicklung der lokale Code mit den entsprechenden Modifikationen genutzt werden soll und z. B. ein ständiges Neubauen des Applikation-Images und damit unnötiger Zeitaufwand verhindert werden soll.

Die Konfiguration des MongoDB-Containers erfolgt über Environment-Variablen und ein *mongo-init.sh*-Skript, welches beim ersten Starten des Containers von dem MongoDB-Server ausgeführt wird. Dieses legt basierend auf Environment-Variablen einen neuen DB-User (cme) an, welcher nur in einer Datenbank mit dem Namen *cme_data* Lese- und Schreibrechte hat.

Das Applikation-Image wird bei jedem Starten der docker-compose-Umgebung frisch gebaut und enthält die CME-Software, welche durch die Tatsache, dass es sich dabei um ein installierbares Python-Package handelt, einfach nur dort installiert werden muss.

Zusätzlich wurde die Deployment-docker-compose-Konfiguration so konfiguriert, dass ein Zugriff auf die Container von außen nicht möglich ist. Dies bedeutet einerseits, dass der DB-Port des MongoDB-Containers nur durch den CME-Container erreichbar ist und andererseits, dass die REST-Endpoints des CME-Containers nur über eine Verbindung von Localhost aus erreichbar sind. Der Zugriff von außen erfolgt über einen zusätzlich konfigurierten nginx-Reverse-Proxy. Dieser hat neben Secure Shell (SSH) den einzigen nach außen hin geöffneten Port

auf dem Server. Somit muss jeder Request durch diesen Reverse-Proxy fließen, bevor er in die docker-compose-Umgebung gelangt.

Datensicherheitsvorfall

Während der initialen Konfiguration der Infrastruktur wurde der interne Kommunikationsport des genutzten MongoDB-Containers aufgrund einer fehlerhaften docker-compose-Konfiguration durch den Docker-Daemon nach außen geöffnet und nicht, wie gedacht, nur lokal. So konnte auf die so für den Rest des Internets frei zugängliche MongoDB von außen zugegriffen werden und diese wurde kompromittiert.

Nachdem diese Situation dem Team bekannt wurde, wurde dies den entsprechenden Parteien gemeldet und das System wurde ein zweites Mal aufgesetzt. Für diese zweite Konfiguration wurde ausgiebig die vorliegende docker-compose-Konfiguration überprüft. Dabei stellten wir fest, dass die Verwendung des *ports*-Arguments ohne die Angabe einer expliziten Internet Protocol (IP), wie es in vielen docker-compose-Tutorials zu finden ist, dazu führt, dass der Docker-Daemon diesen Port mithilfe einer zusätzlichen IP-Tables-Chain nach außen hin öffnet. Aus diesem Grund wurde in der offiziellen Dokumentation als Lösung nach alternativen Möglichkeiten gesucht, Ports freizugeben. Dort wurde einerseits die bereits erwähnte Lösung gefunden, welche durch die explizite Angabe einer „Bind“-IP-Adresse wie z. B. hier „127.0.0.1:9001:9001“ dazu führt, dass nur Anfragen, die über diese Adresse erfolgen, beantwortet werden. Andererseits wurde das *expose*-Argument entdeckt, welches diesen Port nur für alle Docker-Container innerhalb derselben docker-compose-Umgebung freigibt.

Zur vollständigen Beseitigung des Problems wurde schließlich, wie auch schon im vorherigen Abschnitt beschrieben, das *expose*-Argument genutzt, um den MongoDB-Container nur innerhalb der docker-compose-Umgebung zugänglich zu machen. Zusätzlich wurde das Portbinding des CME-Containers, welcher den REST-Service enthält, auf Localhost beschränkt. Die eigentliche Auflösung von außen erfolgt dann über einen nginx-Reverse-Proxy, welcher im Hostsystem unter einem nicht privilegierten Nutzer läuft. So ist jeglicher Zugriff auf die docker-compose-Umgebung von außen nicht mehr möglich und muss durch den Reverse-Proxy.

3.2.6 Schnittstelle für Zugriff auf den Communication Model Extractor

Um den gewünschten Workflow einer Pipeline zu ermöglichen, entschieden wir uns eine API anzubieten, wodurch die Kommunikation zur vorherigen und folgenden Gruppe realisiert wurde.

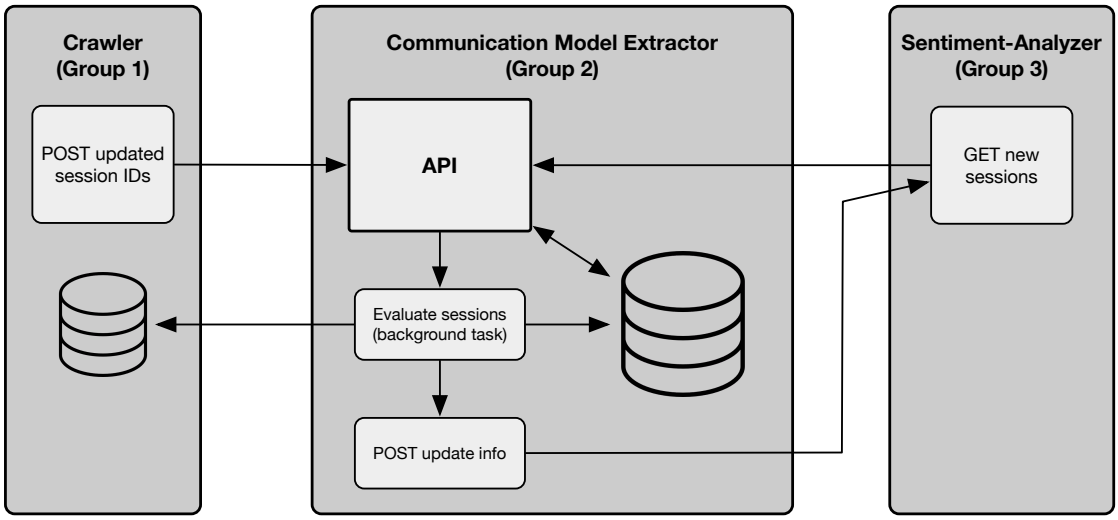

Abbildung 3.2: API-Kommunikationsfluss

Es wurde sich mit Gruppe 1 darauf geeinigt, dass der CME einen Endpunkt anbietet, um die Information zu erhalten, wenn neue Protokolle aus dem Bundestag in deren Datenbank kreiert wurden. Nach dem Erhalt der Zugangsdaten konnte auf die Datenbank der Gruppe 1 zugegriffen werden können. So werden die Protokoll-ID's an `http://infosys2.f4.htw-berlin.de:9001/cme/data/` gesendet und der CME holt sich die gewünschten Protokolle zu einem späteren Zeitpunkt aus deren Datenbank. Danach kann die Auswertung geschehen.

Um der darauffolgenden Gruppe Bescheid zu geben, wird nach dem Prozess der Auswertung ebenfalls eine HTTP Anfrage gesendet, in der über die neuen Protokoll-ID's informiert wird. Die API bietet dafür verschiedene Endpunkte an, um an alle notwendigen Informationen zu gelangen.

Diese Endpunkte sind alle in einem Endpunkt festgehalten, die die API dokumentiert und hier abgerufen werden kann: `http://infosys2.f4.htw-berlin.de:9001/cme/doc/docs`

Demnach enthält der CME folgende Endpunkte:

- /cme/data/sessions/ - damit wird eine Liste aller existierenden Sitzungen zurückgegeben
- /cme/data/session/{session_id} - um eine bestimmte Sitzung mit den ermittelten Interaktionen zu erlangen
- /cme/data/mdb - nützlich, um nach Mitgliedern des Bundestags zu suchen, mithilfe der Query-Parameter mdb_id, speaker_id, forename & surname kann gefiltert werden

- /cme/data/faction - damit können alle existierenden Fraktionen mit jeweiliger ID abgerufen werden

Der Server ist nur über das Hochschule für Technik und Wirtschaft (HTW)-Netzwerk erreichbar. Zusätzlich ist die API mit Basic Authentication geschützt. Dadurch kann nur auf die Daten zugegriffen werden, wenn sich mit Username und Passwort authentifiziert wird.

Es wurden drei Clients, sprich User, angelegt, die auf die API zugreifen dürfen. Je einer für die anderen Gruppen crawler_client & sentiment_client und cme_admin für unser Team, um Testen zu können und bei Bugs der Ursache auf den Grund gehen zu können.

3.3 Fazit

Im Rahmen des Teilprojektes konnte erfolgreich die Extraktion des Kommunikationsmodells basierend auf den „*Open Data*“-XML-Dateien sowie dem JSON-Format von Gruppe 1 umgesetzt werden. Dabei wurden alle in Unterabschnitt 3.1.2 definierten Anforderungen erfolgreich erfüllt.

Der Service bietet Schnittstellen zur vorherigen und nachfolgenden Gruppe sowie automatisierte Verarbeitungsmechanismen (notify). Auch die frühzeitige Bereitstellung von Testdaten für die nachfolgenden Gruppen in einer akzeptierten Kommunikations-Modellierung auf Basis der XML-Rohdaten konnte realisiert werden. Der Großteil des Datenbestands wird analysiert, sodass sowohl Kommentare als auch Redeteile auf Kommunikationen überprüft werden.

Allerdings verwirft die CME-Applikation aufgrund von Inkonsistenzen in der Syntax und Rechtschreibung in den Rohdaten sowie aufgrund von fehlenden kontextbasierten Analyse-Mechanismen, potenzielle Interaktionen. Während in Kommentaren primär durch Abweichungen von Protokoll-Syntax und Rechtschreibfehler Probleme entstanden, so fehlte in den Redeteilen oft die nötige Information, um die Empfänger von Interaktionen eindeutig zu bestimmen. Daher mussten gelegentlich Interaktionen verworfen und teils in Kommentaren inkorrekt genannte Personen als Personenduplikate in den Datenbestand aufgenommen werden.

3.4 Ausblick

Das Ziel der Gruppe „Kommunikationsmodell“ wurde in Funktionalität und Integration zwar erreicht, allerdings wird Verbesserungspotential in der Extraktion von noch mehr Interaktionen und auch in der Eliminierung von Artefakten gesehen.

Insbesondere in den Redeteilen besteht in den mehrfach vorkommenden Nachnamen und mehrdeutigen Fraktionsbezeichnern die Möglichkeit deutlich mehr In-

teraktionen zu extrahieren. Dies könnte z. B. durch kontext-basierte Algorithmen, die eine eindeutige Empfänger-Bestimmung durchführen könnten, realisiert werden. In diesem Zuge wird auch die Ermittlung von False Positives, also Fällen in denen der von uns ermittelte Empfänger eigentlich nur durch indirekte Rede erwähnt wird, mit zunehmender Interaktionsanzahl wichtiger, um zukünftig die korrekte Erkennung von Interaktionen zu gewährleisten.

Die ausgelieferte Deployment-Strategie und die Modulbeschreibungen in dieser Dokumentation sollten bei der Weiterentwicklung des Projekts hilfreich sein.

Kapitel 4

Sentiment Analyse

Gruppenmitglieder: Awad, Laas, Sboui

4.1 Einleitung

Der Begriff *Sentiment* stammt vom lateinischen Wort *sentimentum* ab und bedeutet Empfindung oder Stimmung. In der Sentimentanalyse geht es um die Bestimmung eben jener Stimmung einer Meinungsäußerung. Anwendung findet sie etwa bei Produktbewertungen oder Beiträgen in sozialen Netzwerken. Die Stimmung wird durch die sog. Polarität beschrieben und kann positiv, negativ oder neutral ausfallen. Im Text-Mining wird sie im Intervall $[-1, 1] = \{x \in \mathbb{R} | -1 \leq x \leq 1\}$ angegeben, wobei -1 sehr negativ, 1 sehr positiv und 0 neutral bedeuten.

Grundlegend ist die maschinelle Sentimentanalyse entweder Wortlisten- oder Modell-basiert. Der Wortlisten-Ansatz kann als eine Sammlung von Worten und ihrer Polaritäten verstanden werden, anhand welcher die Stimmung berechnet wird. Dieser Ansatz wird weiter im Kapitel 4.3 beschrieben, da er in dieser Arbeit verwendet wurde.

Bei der Modell-basierten Sentimentanalyse wird ein annotierter Satz-Korpus für das Trainieren einer künstlichen Intelligenz vorausgesetzt. Etwa für Twitter-Beiträge stehen solche Korpusse oder auch bereits trainierte Modelle zur Verfügung. Jedoch ist bei der Verwendung zur Analyse der Sitzungsprotokolle des Bundestages nicht mit zufriedenstellenden Ergebnissen zu rechnen, da sich verwendete Worte und Text- bzw. Satzbau zwischen diesen Domänen deutlich unterscheiden. Eine weitere Erläuterung zur Sprache im Bundestag wird in Kapitel 4.5.1 gegeben.

Ebenfalls werden die Komponenten zur Teilnahme an der Verarbeitungspipeline des Gesamtprojektes im nachfolgenden Kapitel 4.2 beschrieben.

4.2 Datenaustausch

4.2.1 Datenimport

Für das Erhalten von Daten wurde eine Methode implementiert, welche durch Angabe einer Sitzungs-ID die entsprechende Sitzung vom REST-API von Gruppe 2 abfragen kann. Diese Methode gibt jede Sitzung weiter an die in Kapitel 4.4 beschriebene Textverarbeitung und das Ergebnis schließlich an den in Kapitel 4.2.2 beschriebenen Export-Code.

Die Methode wird für zwei Fälle verwendet: Beim Normalfall sendet die Gruppe 2 eine Benachrichtigung mit einer Liste aller neuen Sitzungs-IDs an das REST-API im Code dieser Arbeit. Das REST-API wurde mit der Bibliothek `Flask` [12] entwickelt und besteht aus einer POST-Schnittstelle, welche auf dem von der HTW bereitgestelltem Server unter */notify* erreichbar ist.

Das API wurde unter Zuhilfenahme von `Flask.Blueprints` [13] entwickelt und wird von einem `Waitress`-Webserver [14] bereitgestellt. Für jede Anfrage wird ge-

prüft, ob Content-Type und Payload dem erwarteten JSON-Daten entsprechen. Sollte dies nicht der Fall sein, wird eine entsprechende Rückmeldung an den Sender zurückgegeben. Für jede erhaltene Sitzungs-ID wird die eingangs beschriebene Methode aufgerufen.

Der zweite Fall wurde vor allem im Rahmen der voranschreitenden Entwicklung bei den in der Projektpipeline voranstehenden Gruppen verwendet. Statt auf eine Benachrichtigung zum Anstoßen des Datenimports zu warten, werden stattdessen alle Sitzungs-IDs vom REST-API der Gruppe 2 abgefragt und jede Sitzung vollständig neu importiert. Dieses Vorgehen hat den Vorteil, dass eventuell durch Arbeit am Code verpasste Benachrichtigungen nachgeholt werden und gleichzeitig Änderungen an den Daten übernommen werden können.

4.2.2 Datenexport

Statt Arbeit in ein umfangreiches REST-API für die nachfolgenden Gruppen zu stecken, wurde für diese Arbeit ein reiner MongoDB-Ansatz verwendet. Auf dem zur Verfügung stehenden HTW-Server wurde dafür eine MongoDB Instanz aufgesetzt, welche nur aus dem HTW-Netz erreichbar und zudem nur mit Authentifizierung zugreifbar ist. Jede Sitzung ist hier als eine eigene Collection persistiert. Sobald neue Sitzungen importiert und verarbeitet wurden, werden diese zunächst mit dem MongoDB-Treiber `PyMongo` [15] in die Datenbank geschrieben. Anschließend werden die nachfolgenden Gruppen mit einem POST and die jeweilige REST-Schnittstelle benachrichtigt, dass neue Daten vorliegen. Diese greifen dann mit den zuvor versendeten Zugangsdaten direkt auf die Datenbank zu.

Mit diesem Vorgehen konnte die Komplexität beim Zugriff auf die Ergebnis-Daten gesenkt werden, da es für nahezu alle gängigen Programmiersprachen einen intuitiven MongoDB-Treiber gibt.

4.3 Wortliste

Wie bereits in der Einleitung beschrieben, handelt es sich bei Wortlisten in der Sentimentanalyse um eine Liste von Worten und ihrer jeweiligen Polarität. Bei der Analyse eines Textes wird wortweise ein Abgleich mit dieser Liste durchgeführt und die Wort-Polarität bei einer Übereinstimmung für die Berechnung des Sentiments verwendet. Unterschiedliche Berechnungsformeln sind dabei denkbar und werden in Kapitel 4.4.3 weiter besprochen. Für diese Arbeit wurden Wortlisten verschiedener Institutionen kombiniert und anschließend mit Synonymen erweitert. Ebenfalls wurde eine eigene Bundestags-Wortliste erstellt.

Die Interest Group on German Sentiment Analysis (IGGSA) stellt eine umfangreiche Liste an Publikationen und Ressourcen zu Sentimentanalysen in der

deutschen Sprache zur Verfügung [16]. Hier wurden alle Referenzen auf die in dieser Arbeit verwendeten Quell-Wortlisten gesammelt. Für die Zusammenführung dieser Wortlisten wurden zunächst zwei Herangehensweisen evaluiert: Zum einen war die Erstellung eines eigenständigen Codes denkbar, welcher einmalig die Daten aus allen Quellen einliest, zusammenfasst und eine Ergebnis-Datei ausgibt. Diese Datei wäre eine Ressource für den eigentlichen Analyse-Code. Zum anderen könnte die soeben beschriebene Funktionalität jedoch auch direkt im Analyse-Code integriert und bei jedem Programmstart ausgeführt werden. Die kombinierte Wortliste würde somit nicht auf die Festplatte geschrieben, sondern im Arbeitsspeicher verbleiben, solange der Analyse-Code läuft. Wenngleich der zuerst beschriebene Ansatz offensichtlich weniger Rechenzeitaufwändig ist, wurde für diese Arbeit der zweite Ansatz gewählt. Dies wird vor allem mit Rechtsunsicherheiten bei der Arbeit mit den verschiedenen Lizenzen der Quelldateien begründet. Gleichzeitig arbeitet der Analyse-Code damit jederzeit mit dem aktuellen Stand der Quell-Wortlisten. Das Aufbauen der Wortliste dauert wenige Minuten.

Für diese Arbeit wurden die folgenden Quell-Wortlisten verwendet:

- *SentimentWortschatz* (SentiWS) aus „SentiWS - a Publicly Available German-language Resource for Sentiment Analysis“(Universität Leipzig) [17]
- *Multi-Domain Sentiment Lexicon for German* (Hochschule Darmstadt) [18]
- *German Polarity Lexicon* aus „Evaluation and extension of a polarity lexicon for German“(Universität Zürich) [19]
- *morphcomp* aus „Evaluating the morphological compositionality of polarity“(Leibniz-Institut für Deutsche Sprache, Universität des Saarlandes) [20]

Aus diesen Quellen ergibt sich eine Wortliste mit etwa 14.000 einzigartigen Worten. Die Worte werden dabei mit der in Kapitel 4.4 beschriebenen Textverarbeitungspipeline lemmatisiert, also auf die Grundform zurückgeführt. Bei Überschneidungen wird ein Mittelwert über alle Polaritäts-Werte eines Wortes gebildet.

Mithilfe des *Open German WordNet* (OdeNet) [21] werden zu jedem Wort Synonyme gesammelt und ebenfalls der Wortliste hinzugefügt. Der Zugriff auf OdeNet geschieht dabei mit der python Bibliothek `WN` [22]. Die Verwendung dieser Bibliothek ist trivial, weshalb an dieser Stelle keine weitere Erläuterung gegeben wird. Die Wortliste wird durch das Hinzufügen von Synonymen um weitere rund 2.000 Worte erweitert.

Wie bereits in der Einleitung erwähnt, wurde zudem eine eigene Bundes-tags-Wortliste erstellt. Hierzu wurden die 15.000 am häufigsten in den Sitzungsprotokollen vorkommenden Worte erfasst und analysiert. Dabei wurden jedoch nur jene Worte betrachtet, welche nicht bereits in der kombinierten Wortliste vorkommen.

Ausgewählt wurden nur eindeutig positive oder negative Worte wie *angemessen*, *Bullshit*, *Fehlentscheidung*, *Milchmädchenrechnung*, *Realitätsverweigerung*, *Totalausfall* oder *Verunglimpfung*. Insgesamt umfasst die Bundestags-Wortliste 217 Worte.

Im Code wird der Zugriff auf die Wortliste mit der zentralen Klasse `Lexicon` realisiert. Bei der Instanziierung der Klasse werden, wie im Vorherigen beschrieben, alle Wortlisten gesammelt, zusammengeführt und erweitert. Anschließend stellt die Klasse ein `Dictionary` bereit, in welchem die Schlüssel die Worte und die Werte die Wort-Polarität angeben.

4.4 Textverarbeitung

Für die Textverarbeitung nutzt der Code dieser Arbeit vollständig die Funktionalitäten und Strukturen von `spaCy` [23]. Die `spaCy` Bibliothek stellt eine Textverarbeitungspipeline bereit, welche mithilfe eines vortrainierten Modells funktioniert. Für die deutsche Sprache wurde ein solches Modell mit dem TIGER Korpus der Universität Stuttgart trainiert. Der Korpus umfasst ca. 50.000 Sätze aus Texten der Frankfurter Rundschau. Die `spaCy`-Pipeline besteht aus den folgenden Komponenten:

- Tokenisierung (Text- und Satzzerteilung)
- POS-Tagging (Wortartenerkennung)
- Dependency-Parsing (Wortabhängigkeitenerkennung)
- Lemmatisierung (Wortgrundformermittlung)
- Entity-Recognition (Eigennamenerkennung)

Das Ergebnis der Pipeline ist ein `Doc`-Objekt, welches das Ergebnis des gesamten in die Pipeline gegebenen Textes umfasst. Einzelne Sätze innerhalb des `Doc`-Objektes werden mit `Span`-Objekten beschrieben und einzelne Worte mit `Token`-Objekten. Die Objekttypen besitzen jeweils eigene Methoden und Erweiterungsmöglichkeiten in Form von sog. *extension attributes*.

Sehr einfach und gleichzeitig umfangreich kann die `spaCy`-Pipeline an die eigenen Bedürfnisse angepasst werden. So wurde die Berechnung des Sentiments eines Textes direkt an die Komponenten der Standart-Pipeline angehängt. Logisch unterteilt sich diese in die Komponenten:

- Negations-Erkennung (siehe 4.4.1)

- Verstärkungs-Erkennung (siehe 4.4.2)
- Polaritätsberechnung (siehe 4.4.3)

Basis für das nachfolgende Kapitel 4.4.1 ist dabei das Ergebnis des Dependency-Parsers von `spaCy`. Dieser bestimmt die Abhängigkeiten zwischen den einzelnen Elementen eines Satzes. Diese Funktion gehört dem Themenbereich der Dependenzgrammatik an, in welcher gerichtete Beziehungen zwischen den Worten eines Satzes beschreibt werden. Ein Wort kann einen Vorgänger (Regent) und mehrere Nachfolger (Dependenten) besitzen. Die Gesamtheit der Beziehungen eines Satzes wird auch Abhängigkeitsbaum genannt.

Zur Verdeutlichung soll die Abbildung 4.1 dienen, welche mithilfe von `spaCy.displaCy` erstellt wurde. Zu sehen ist hier der Abhängigkeitsbaum des Satzes *„Das ist doch fachlich Quatsch hoch sechs!“*. An den gerichteten Kanten befindet sich die Angabe des Satzgliedes, unterhalb der Worte die Angabe der Wortart.

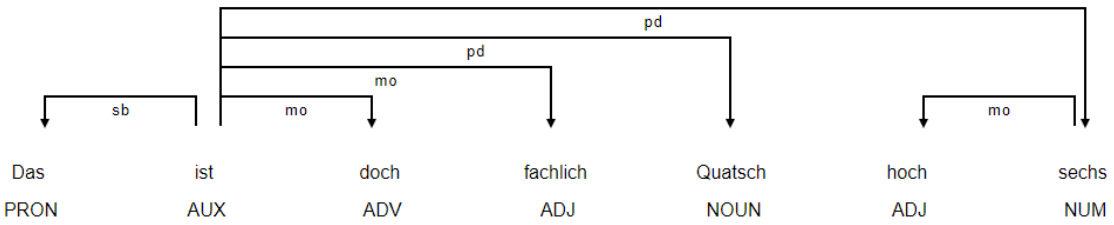

Abbildung 4.1: Visualisierung der Wortabhängigkeiten (Zitat von Steffi Lemke MdB, 208. Sitzung, 10.02.2021)

4.4.1 Negations-Erkennung

Negation, also Ablehnung, Verneinung oder Aufhebung, hat einen erheblichen Einfluss auf das Ergebnis und damit die Korrektheit der Sentimentanalyse, weshalb in dieser Arbeit ein besonderes Augenmerk auf ihre Erkennung gelegt wurde. In *Negation Modeling for German Polarity Classification* [24] präsentieren Forscher der Universität des Saarlandes und des Leibniz-Institut für Deutsche Sprache hierfür einen regelbasierten Ansatz.

Sie definieren unterschiedliche Negationstypen und ihre jeweilige Reichweite. So beeinflussen etwa negierende Adverbien oder Indefinitpronomen wie *nie* den gesamten Satz, wohingegen das Partikel *nicht* lediglich seinen Vorgänger negiert. Die Tabelle 4.1 führt alle in dieser Arbeit implementierten Negationsregeln auf. Die Nutzung eines Abhängigkeitsbaumes, wie von `spaCy` ermittelt, ist dabei unerlässlich. Denn wie schon im vorherigen Kapitel angesprochen, ist etwa mit dem

Tabelle 4.1: Implementierte Negations-Regeln aus [24]

Negationstyp	Reichweite	Beispielworte
Partikel	Vorgänger (Regent)	nicht
Präpositionen	Nachfolger (Dependent)	ohne, gegen
Adverbien, Indefinitpronomen	Satz	nie, kein, kaum
Nomen	Genitiv, Präpositionalobjekt	Abschaffung, Zerstörung
Verben	Objekt, Subjekt	enden, sinken, lindern

Vorgänger eines Wortes nicht das in der Satzabfolge voranstehende Wort gemeint, sondern vielmehr der semantische Regent.

Angemerkt sei an dieser Stelle, dass es nicht möglich war, alle Regeln aus [24] zu implementieren, da der von den Forschern genutzte Dependency-Parser umfangreichere Ergebnisse liefert, als jener von `spaCy`.

Eine Liste mit Negationsworten und dem jeweiligen Negationstyp wurde [19] entnommen und ist ebenfalls über die Klasse `Lexicon` zugreifbar. Die Klasse stellt ein `Dictionary` bereit, in welchem die Schlüssel die Negationsworte und die Werte eine Liste der Reichweiten sind.

Wenn in einem Text ein Negationswort auftritt, werden alle implementierten Regeln geprüft. Sollte es zu einem Treffer kommen, etwa wenn das Wort *nicht* auftritt (siehe Abb. 4.2) und es einen Vorgänger gibt, werden alle Worte in Reichweite des Negationswortes negiert. Dies geschieht, indem für die jeweiligen Worte, welche wie in 4.4 beschrieben `Token`-Objekte sind, ein eigenes Attribut mit der Bezeichnung `negated` auf `True` gesetzt wird. In der Polaritätsberechnung wird wortweise auf dieses Attribut geprüft und die Wort-Polarität bei einer Negierung mit -1 multipliziert.

4.4.2 Verstärkungs-Erkennung

Als *Verstärker* werden sog. Gradpartikel (z.B. sehr, besonders, viel) verstanden, welche direkt vor Adjektiven oder Adverbien in einem Satz auftreten. Sie verstärken ihren Nachfolger, was wiederum in der Polaritätsberechnung berücksichtigt werden soll.

Aus diesem Grund wird eine Liste mit verstärkenden Gradpartikeln, welche ebenfalls aus [19] bezogen werden, eingelesen und über die `Lexicon` Klasse bereitgestellt.

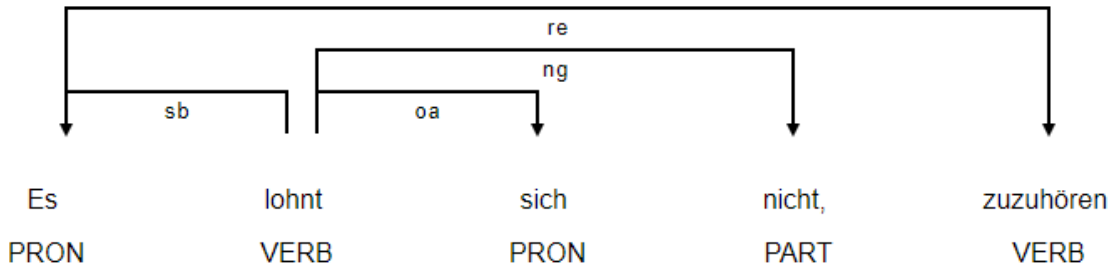

Abbildung 4.2: Beispielsatz mit Patikel-Negation (Zitat von Stephan Brandner MdB, 207. Sitzung, 29.01.2021)

Ebenso wie in Kapitel 4.4.1 bereits für die Negation beschrieben, wird ein eigenes Attribut zur Signalisierung einer Verstärkung definiert und im entsprechenden Fall auf True gesetzt. Bei der Polaritätsberechnung wird bei einer erkannten Verstärkung die Wort-Polarität mit 1,5 multipliziert [25].

In Abbildung 4.3 wird ein Beispiel für das Auftreten eines verstärkenden Gradpartikels gegeben. Hier verstärkt das Wort *sehr* das negative Wort *spät*, womit die errechnete Polarität stärker negativ ausfällt als ohne die Verstärkungs-Erkennung.

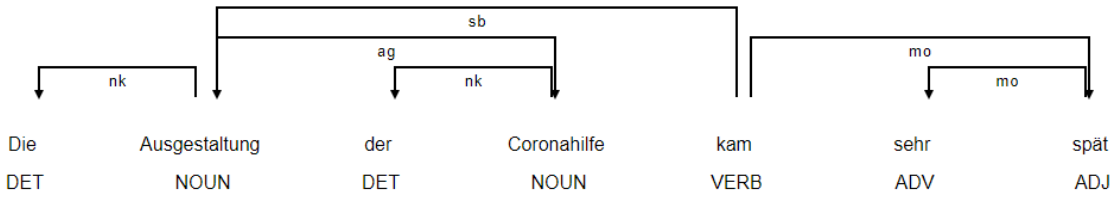

Abbildung 4.3: Beispielsatz mit Gradpartikel-Verstärkung (Zitat von Katja Hessel MdB, 206. Sitzung, 28.01.2021)

4.4.3 Polaritätsberechnung

Für die abschließende Berechnung der Polarität sind verschiedene Formeln denkbar. Sie sollten anhand von Textcharakteristika wie z.B. der Satz- oder Textlänge gewählt werden.

Bei der Verwendung einer satzbasierten Polaritätsberechnung, bei welcher alle Polaritäten erst addiert und die Summe anschließend durch die Anzahl der Worte dividiert wird, kann ein unerwünschtes Phänomen auftreten: Längere Sätze erhaltenen ein stärker polarisiertes Ergebnis als vergleichbare kurze Sätze.

Dies ist mit einer im Schnitt höheren Dichte an Worten mit Polaritäts-Wert in längeren Sätzen zu erklären. Um diesem Problem entgegen zu wirken, wurde in

dieser Arbeit eine Min-Max-Skalierung (siehe Formeln 1 - 3) auf Dokumentebene implementiert [25].

Die Entscheidung, diese Normalisierung anhand der Länge des gesamten Textes durchzuführen, wurde aufgrund der Charakteristika der zu analysierenden Interaktionen getroffen. Denn diese bestehen zu einem überwiegenden Teil aus einem einzelnen Satz von jedoch sehr unterschiedlicher Länge.

$$p' = \frac{p+1}{text.len+1} \text{ für p} > 0 \tag{4.1}$$

$$p' = \frac{p-1}{text.len+1} \text{ für p} < 0 \tag{4.2}$$

$$p' = p \text{ für p} = 0 \tag{4.3}$$

4.5 Evaluierung

4.5.1 Sprache im Bundestag

Während der Entwicklung dieser Arbeit und den regelmäßig angestellten Zwischentests wurde ersichtlich, dass sich die Sprache im Bundestag etwa von jener in sozialen Netzwerken oder Produktbewertungen unterschiedet. Ein Interaktionstext setzt sich sowohl aus einzelnen, langen und komplexe Sätze zusammen, als auch aus einzelnen Ausrufen wie *„eieiei!“* zusammen.

Aus diesem Grund wurde die in 4.4.3 beschriebene Normalisierung verwendet, da herkömmliche Berechnungsformeln zunächst widersprüchliche Ergebnisse lieferten. Gleichzeitig verbesserte die manuelle Durchsicht der häufigsten 15.000 Worte und Anfertigung einer eigenen Bundestags-Wortliste das Ergebnis deutlich. Viele der regelmäßig in Bundestagssitzungen verwendeten Worte gehören zur *Politik-Domäne* und treten deshalb nicht in den Quell-Wortlisten auf. Hier wird erwartet, dass eine noch umfangreichere Bundestags-Wortliste einen weiteren positiven Einfluss auf das Endergebnis hätte. Aus Zeit- sowie Kompetenzgründen wurde diese Liste jedoch nicht erweitert.

4.5.2 Ironie und Sarkasmus

Ebenso wie die im vorherigen besprochene *Politiksprache*, treten auch Ironie und Sarkasmus vermehrt in den Sitzungen des Bundestages auf. Einfach umrissen, handelt es sich dabei um ein Stilmittel, bei dem das Gegenteil vom Gesagten gemeint ist. Sarkasmus ist dabei eine verstärkte Form der Ironie und kann auch als ein Angriff verstanden werden.

Selbst für den menschlichen Leser ist allein am geschriebenen Text nicht immer ersichtlich, ob eine Aussage ironisch gemeint ist. Vielmehr wird für die richtige Deutung die Stimmlage, Gestik und Mimik der sprechenden Person benötigt.

Ironie und Sarkasmus könne also erst recht nicht mit dem in dieser Arbeit verwendeten Wortlisten-Ansatz erkannt werden, womit ein unbekannter Teil der Interaktionen im Ergebnis die falsche Polarität besitzt. Gleichwohl gibt es Ansätze aus dem Bereich des maschinellen Lernens, welche dieses Problem behandeln. Jedoch setzen diese einen entsprechend annotierten Korpus vorraus und stammen zudem aus dem Bereich der sozialen Netzwerke, in welchen etwa mit *Hashtags* die Ironie bereits vom Autor markiert wurde.

4.5.3 Fazit

Trotzt der soeben beschriebenen Fehlerquellen, wurde dennoch ein gelungenes Ergebnis erzielt: Es wurde ein solider und erweiterbarer Algorithmus zur Sentimentananlyse von Texten geschaffen, welcher auf den bekanntesten Bibliotheken und Techniken der Textverarbeitung beruht. Dieser kann als Basis für weitere Anstrengungen bei der Sentimentanalyse von politischen Texten dienen.

Eine Deutung der bisherigen Ergebnisse durch Politikwissenschaftler oder anderweitig in diesem Bereich kompetente Personen wird dabei empfohlen, um die beschriebenen Schwachstellen entsprechend zu behandeln oder neue zu identifizieren.

Der Programmcode ist zudem vollständig und abgesichert in die Projektpipeline eingebunden. Er erweitert seine Ergebnis-Datenbank automatisch um neue Sitzungen und benachrichtigt anschließend nachfolgende Gruppen.

Kapitel 5

Analyse der Interaktion zwischen Abgeordneten

Gruppenmitglieder: Florian Thom, Jennifer Vormann, Rico Stucke

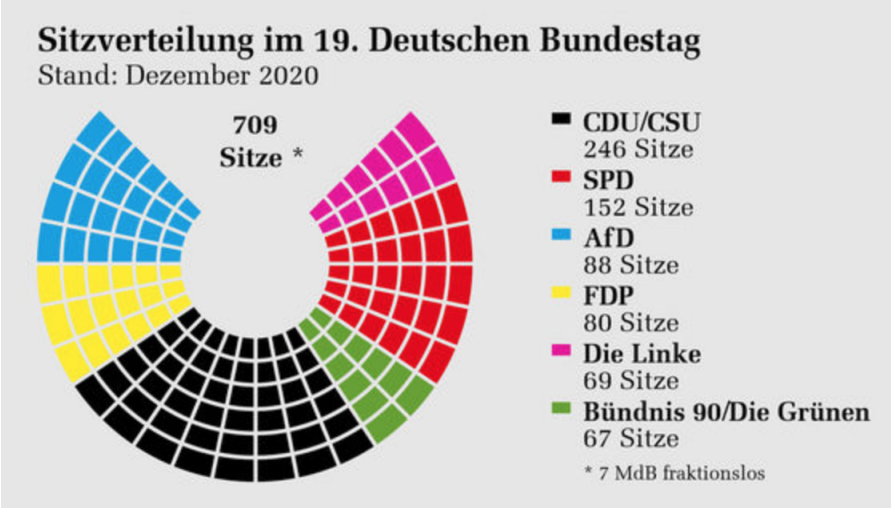

Abbildung 5.1: Sitzverteilung im 19. Deutschen Bundestag [26]

5.1 Aufgabe Team 4

(Autorin: Jennifer Vormann)

Team
Rico Stucke | Florian Thom | Jennifer Vormann

Thema
Graph-basiertes Informationssystem zur Analyse sozialer Interaktion im Deutschen Bundestag

Aufgabe von Team 4
Unsere Aufgabe bestand darin, die Beziehungen zwischen Abgeordneten darzustellen. Die Beziehungen ausgehend von Abgeordneten zu einer Fraktion und von einer Fraktion zu einem Abgeordneten wurden ebenso einbezogen und betrachtet. Kommentare von einer Fraktion zu einer anderen wurden bei dieser Teilaufgabe außen vor gelassen, da dies Aufgabe von Gruppe 5 war.
In der Praxis bedeutete das für uns Daten zwischen unterschiedlichen Datenbanken und Servern zu verarbeiten und zu übertragen. Ziel war es, eine Graphdatenbank zu haben, in der die Interaktionen zwischen den Abgeordneten während der Bundestagssitzungen ersichtlich sind und abgerufen werden können. Es wurden alle Sitzungstage der 19. Wahlperiode betrachtet. Aktuell fanden in dieser Wahlperiode über 200 Sitzungstage statt. Wechsel von Abgeordneten in eine andere

Fraktion wurden vorerst außen vor gelassen. Das genaue Vorgehen zur Erreichung dieses Ziels wird im nachfolgenden Kapitel erläutert. Der Hintergrund ist, herauszufinden, ob der Ton im Bundestag tatsächlich rauer geworden ist. Für die Analyse liegen die entsprechenden Sitzungsprotokolle als XML-Dateien vor. Uns wurde von Team 3 Zugang zu einer MongoDB mit den bereits bereits gespeicherten Daten gewährt.

5.2 Vorgehen

(Autorin: Jennifer Vormann)

5.2.1 Absprachen

Im Rahmen der Plenarsitzungen wurde mit allen Gruppen gemeinsam das Vorgehen besprochen. Dies geschah im Rahmen der wöchentlichen Meetings. Zusätzlich fanden diverse Absprachen mit Team 2 und 3 statt. Es musste geklärt werden, wann wir die ersten Daten erhalten und in welcher Form. Mit Team 5 musste abgesprochen werden, ob wir den gleichen Server und/oder die gleiche Datenbank benutzen wollen. Hier wurde sich in beiden Punkten dagegen entschieden. Die Kommunikation mit Team 7 fand ebenfalls ergänzend statt, um die Weiterleitung der Daten zu besprechen.

5.2.2 Planung & Setup

Zu Beginn des Projektes haben wir ein GitHub Repository erstellt und den virtuellen Server aufgesetzt. Wir haben einen Zeitplan angefertigt und eine Planungspräsentation vorbereitet, die auch anschließend in der Plenarsitzung gehalten wurde. Im Team haben wir uns gemeinsam auf ein Datenbankschema festgelegt. Das Datenbankschema wird im Unterkapitel zum Entwurf näher erläutert.

Nachfolgend verständigten wir uns im Team und auch mit Gruppe 5 zu den Technologien und nahmen die entsprechenden Installationen vor. Wir entschieden uns für die Graphdatenbank Neo4j, um die Beziehungen zwischen den Abgeordneten gut visualisieren zu können. Eine weitere Intention war die Lust Neo4j und Cypher neu zu erlernen. Beim Skript entschieden wir uns für ein Python Skript. Das Skript, welches zu implementieren war, sollte eine Verbindung zu den Datenbanken (MongoDB, Neo4j) aufbauen und die Daten einzeln von der MongoDB einlesen. Anschließend sollte es in der Lage sein, die Daten umzuformen - in Nodes, Properties & Relationships. Als letzter Schritt sollte die Sicherung in die Graphdatenbank Neo4j realisiert werden, um die Daten Gruppe 7 zur Verfügung stellen zu können.

Zum Ende unseres Projektes haben die gesamte Umsetzung inklusive Ergebnissen und Ausblick in dieser Dokumentation zusammengetragen. Die Abschlusspräsentation wurde erstellt und vor dem Kurs gehalten.

5.3 Entwurf

(Autor: Florian Thom)

Mit diesem Unterkapitel werden Hintergrundwissen, Überlegungen und Prozesse zur Umwandlung von unstrukturierten Daten aus einer NoSQL-Datenbank in ein graphenoptimiertes Schema zur Darstellung gegebener Interaktionen zwischen Abgeordneten des Bundestages präsentiert.

5.3.1 Überblick

Diese Arbeit setzt ein Basisverständnis von Graphdatenbanksystemen voraus. Dementsprechend werden zu besagten Graphdatenbanken keine Begriffsbestimmungen oder weitere Erläuterungen vermerkt.
Ein Graphdatenbankschema ist abhängig von verschiedenen Faktoren. Innerhalb dieser Arbeit wird sich dazu entschlossen auf einen „Labeled Property Graph“ (LPG), statt auf einen Graphen nach dem „Resource Description Framework“ (RDF) zu setzen. Ein Hauptgrund dafür ist, dass man hier versucht sich innerhalb eines PoC zu orientieren.
Bezüglich des LPGs wird sich für eine Graphdatenbank des Unternehmens „Neo Technology“ mit dem Namen Neo4j entschieden. Innerhalb der LPGs ist Neo4j eine der meist verwendeten Datenbanken. Somit scheint die Verwendung, mit dem angebotenen Support durch Schnittstellen zu verschiedenen Sprachen, angemessen.

5.3.2 Graphdatenbank: Neo4j

Einleitend ist festzustellen, dass Neo4j einen „native graph storage“ umsetzt und eine sogenannte „index-free adjacency“ Datenbank ist [27]. Das bedeutet einerseits, dass der Datenspeicher speziell für Graphen angepasst ist. Neo4j verwendet dafür mehrere sogenannte „stores“ [28]. Beispielsweise existiert ein „store“ individuell für alle Knoten oder auch für alle Relationships. Durch den Term „index-free adjacency“ wird nun andererseits beschrieben, dass innerhalb dieser stores direkte physische Pointer, beispielsweise zwischen den Knoten, existieren und diese somit aufeinander zeigen. Im Gegensatz dazu zeigen relationale Datenbanken logisch direkt oder indirekt über IDs aufeinander. Etwaige Optimierungen des Graphen schlagen sich somit direkt auf die Performance des Systems aus.

Das konkrete Speichern ist mit der Darstellung 5.2 veranschaulicht. Zu sehen sind ein Node und eine Relationship, sowie eine Andeutung deren interner Speichereinheiten. Wichtig ist an dieser Darstellung, dass Nodes ihre Relationships zweifach

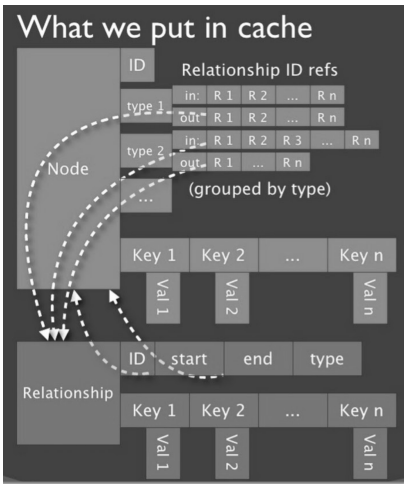

Abbildung 5.2: Datenorganisation in Graphdatenbanken - Beispiel Neo4j [29]

gruppiert speichern [30]. Einerseits werden die Relationships gruppiert nach ihrem Namen (auch Typ) gespeichert. Andererseits wird innerhalb jeder dieser Gruppen differenziert zwischen eingehenden- und ausgehenden Relationships. Durch diese zweifache Gruppierung sollten unter anderem Performanceeinbußen durch Inselbildung nach Typ der Relation oder nach Richtung der Relation vernachlässigbar sein. So muss beispielsweise nicht durch alle eingehenden Relationships iteriert werden, wenn man lediglich nach Ausgehenden sucht.

5.3.3 Graphdatenbankschema: Entwurf

Der Entwurf eines Graphdatenbankschemas ist von einigen Faktoren, u.a. Technischen und Inhaltlichen, abhängig. Diese werden folgend hervorgehoben und definieren eine Art Ausgangslage.
Die Datenbasis ist insofern für die aktuelle Aufgabenstellung wichtig, als das ohne Verständnis für die bestehenden Datenfelder, das Modellieren eines neuen Modells mit neuen Datenfeldern schwierig ist. Ein Überblick über das Schema wird in der Ausarbeitung von Gruppe 2 gegeben. Außerdem ist die These des Projektes zu beachten. Es ist somit darauf zu achten, dass gerade Fragestellungen, die eben in Richtung dieser These gehen, angemessen gut abfragbar sein müssen.
Abschließend werden die konkreten Fragestellungen, die an die Graphdatenbank gestellt werden sollen mit den entsprechenden Verantwortlichen ermittelt. Zum

Zeitpunkt der Ermittlung, soll beispielsweise zu einigen Eigenschaften ein „Score“ berechnet werden, wie zum Sender, die viele Nachrichten mit positivem Sentiment verschicken [31]. Diese „Scores“ sollen (teilweise) für jeweilige Zeitabschnitte berechnet werden, beispielsweise pro Monat. Dies ist insofern relevant, als dass gerade zeitliche Abschnitte vermutlich eine entscheidende Rolle einnehmen.
Ein erster, eher umfangreicher, Entwurf ist in Abbildung 5.3 zu sehen. Dieser und folgende Entwürfe wurden nach eigenem Ermessen strukturiert, wobei sich teilweise an Roy-Hubara et al. [32] orientiert wurde. Der Grafik sind besonders drei

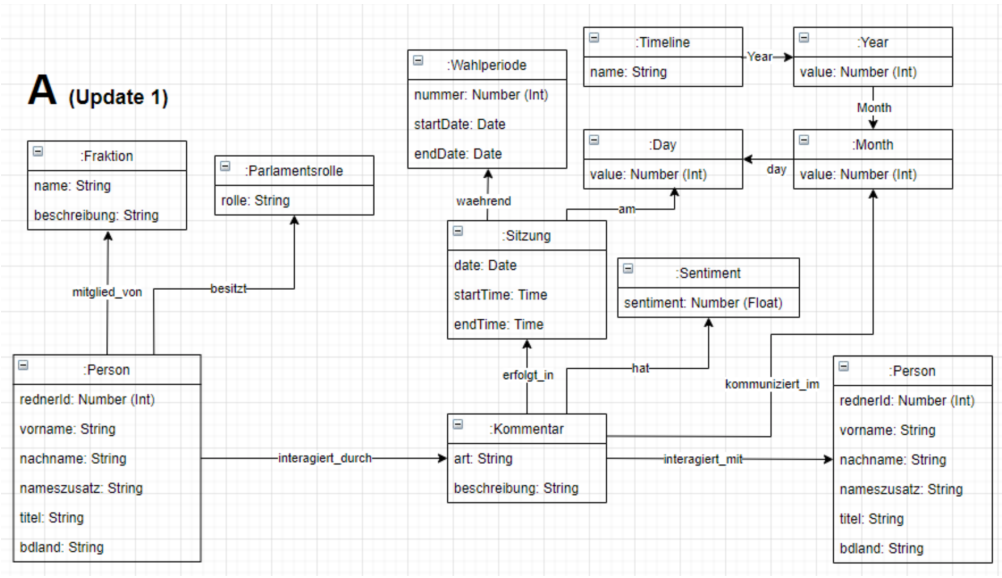

Abbildung 5.3: Initialer Entwurf Graphdatenbankschema

Kernüberlegungen zu entnehmen. Innerhalb der Hauptstruktur hat eine Person eine oder mehrere Fraktionen und eine oder mehrere Parlamentsrollen. Die zweite Kernüberlegung ist die Einführung eines sogenannten „Timeline trees“ (TLT) [27] der mit der Sitzung und den jeweiligen Kommentaren verbunden ist. Er verfügt über die Stufen „:Year“, „:Month“ und „:Day“. Grund für die Integration ist das Wissen über die Wichtigkeit des Faktors Zeit für die Folgeanalysen. Ohne besagten TLT wäre eine Laufzeit von O(N) zu erreichen. Mit Wissen zu TLT könnte man diese Struktur ausnutzen, um die Zugriffszeit für den Erhalt einer gefilterten Menge deutlich zu erhöhen. Beispielsweise könnte man den Zugriff auf alle Kommentare des Monats Februar im Jahr 2020 von O(N) auf in etwa O(2) durch den Zugriff 2020:Year → Februar:Month optimieren.
Eine dritte Kernüberlegung besteht darin, das Sentiment als externen Knoten aus dem Knoten Kommentar zu extrahieren. Wenn man das eher stetige Sentiment (z.B. 0,34216754) in ein eher diskretes Sentiment (z.B. 0,34) transformiert, ist ein

Vorteil bei dem Filtern nach einem bestimmten Sentiment erkennbar (Das Sentiment bewegt sich im Bereich [-1.0, 1.0]). Da diese Transformation bei der Umwandlung der Daten aus der NoSQL-Datenbank in die Neo4j sehr gut möglich ist, sollte diese Überlegung nicht unbemerkt gelassen werden. Da hier beispielsweise nun auf zwei Dezimalstellen nach dem Komma gerundet wurde und das Sentiment sich im Bereich [-1.0, 1.0] bewegt, existieren so 200 verschiedene Gruppen, beziehungsweise demzufolge 200 verschiedene Knotentypen. Der Vorteil wird an einem Beispiel demonstriert. Möchte man eine Menge von Kommentaren erhalten, deren Sentiment positiv ist, müsste man bisher über alle Kommentare iterieren (ca. 70000-250000). Mit der soeben vorgestellten Methode würden sich diese Iterationen auf 100 reduzieren (alle 100 positiven Sentiment-Knoten z.B. 0.01, 0.02, 0.03, ...). Der Basisentwurf ist in Abbildung 5.4 dargestellt. Innerhalb der Hauptstruk-

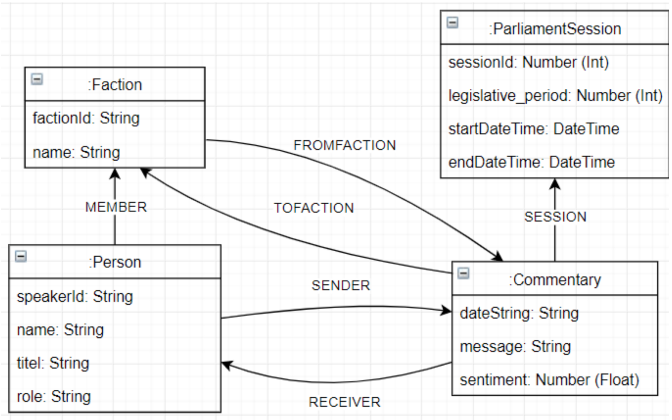

Abbildung 5.4: Datenbankschema

tur ist erkennbar, dass nun zwischen Knoten mit dem Label „:Commentary“ und „:Faction“ eine eingehende- und eine ausgehende Relationship hinzugefügt wurde. Der Grund dafür ist, dass nun teilweise Interaktionen zwischen einem Abgeordneten (einer Person) und einer Partei als Gesamtes identifiziert wurden. Darüber hinaus wurde auf den TLT verzichtet. Grund dafür war, dass dieser anscheinend schwieriger umzusetzen ist.

5.4 Umsetzung

(Autor: Rico Stucke)

5.4.1 Lesen aus MongoDB

Zunächst haben wir uns mit dem Empfang der Daten aus der MongoDB von Team 3 beschäftigt. Für die Abfrage der Daten aus der MongoDB benutzen wir die pymongo Library von Python. Der Client öffnet eine Verbindung zur Datenbank von Gruppe 3 und liest alle Collections, die sich in der Datenbank befinden. Jede Collection entspricht dabei einem Sitzungstag. Im nächsten Schritt aggregieren wir alle ausgelesenen Werte, formen diese um und entfernen eventuelle Dopplungen, die vor allem bei wiederkehrenden Personen oder Fraktionen auftreten. Am Ende dieses Prozesses entstehen Dictionaries, die alle einzigartigen Personen und Fraktionen, sowie Sitzungstage und Kommentare, beinhalten.

5.4.2 Nodes & Relations

Mit der neomodel Library, die als Object Graph Mapper dient, können Klassen definiert werden, die dann von neomodel als Knoten in Neo4j übernommen werden. Außerdem können an diesen Klassen auch die Beziehungen zwischen den Klassen definiert werden. Die entstandenen Dictionaries dienen als Datengrundlage für die Knotenklassen in neomodel. Abbildung 5 zeigt beispielhaft eine Klasse aus dem

```
class Faction(StructuredNode):
    factionId = StringProperty()
    name = StringProperty()
    madeComment = RelationshipTo('Commentary', 'FROMFACTION')
    recievedComment = RelationshipFrom('Commentary', 'TOFACTION')
```

Abbildung 5.5: Beispiel einer neomodel Klasse

Programm, um zu verdeutlichen wie neomodel verwendet wird, um einen Node in der Datenbank zu definieren. Durch die Implementierung von StructuredNode wird neomodel mitgeteilt, dass eine Klasse ein Node in der Datenbank sein soll und über die RelationshipTo und RelationshipFrom Funktionen können die Relationen definiert werden, die von diesem Knoten ausgehen oder eingehen.

5.4.3 Neo4j DB Connection

Die Python library neomodel bietet für das Erzeugen von Nodes und Relations in Neo4j ein einfaches Interface, dass durch seine StructuredNode-Klasse implementiert wird. Dadurch wird es möglich einen Node, der durch eine Klasse, die StructuredNode implementiert, dargestellt wird, über zwei Funktionen in der Datenbank zu speichern und Relationen mit anderen Node-Klassen aufzubauen.
Das Speichern Erfolg durch die von StructuredNode geerbte Methode save(). Hierbei sind keine weiteren Angaben nötig. Diese Methode muss nur vom jeweilig in-

stanziierten Objekt aufgerufen werden, um es in die Datenbank zu übertragen. Für die Herstellung einer Relation muss die connect()-Methode aufgerufen werden. Wichtig hierbei ist, dass die Klasse ein Attribut mit dem Namen der Relation implementieren muss. Der Funktionsaufruf erfolgt dann in dem folgenden Schema:

ObjektA.relation_name.connect(ObjektB)

5.4.4 Laufzeitverbesserungen

In seiner ersten Iteration hat das Programm für die Übertragung einer Collection aus MongoDB in die Neo4jDB rund 5 Minuten gebraucht. Da eine Collection aber nur einen Sitzungstag umfasst, wurde schnell klar, dass Verbesserungen an der Laufzeit nötig sind. Durch Änderungen am Code konnte die Geschwindigkeit gesteigert werden indem nicht mehr für jede Datenbankaktion eine Transaktion verwendet wird, sondern mehrere Aktionen gebündelt werden in einer Transaktion.
Bei einem Test mit einer lokalen Neo4j Datenbank, die außerhalb des HTW-Netzes lief, konnte auch beobachtet werden, dass das Programm wesentlich schneller läuft. Hierbei handelte es sich um Differenzen von 8 Schreibaktionen in der Datenbank pro Sekunde, wenn das Programm von einem privaten Rechner gestartet wurde, der über einen Tunnel mit dem HTW-Netz verbunden war, zu über 200 Schreibaktionen wenn das Programm auf dem selben Rechner lief, der auch die Datenbank gehostet hat. Diese Beobachtung führte dann dazu, dass das Programm um eine einfache REST-API und einen simplen Flask-Server erweitert wurde. Dies ermöglichte es das Programm direkt auf der Hardware laufen zu lassen, die auch die Neo4j Datenbank enthält im HTW-Netz. Durch diese Änderungen gelang es uns die Laufzeit von ursprünglich geschätzten 13 Stunden auf rund 25 Minuten zu reduzieren.

5.5 Ergebnisse

(Autoren: Jennifer Vormann | Rico Stucke)

In seiner jetzigen Form erzeugt das Skript die Neo4j Datenbank in 25 Minuten. Dabei werden rund 278000 Knoten und 830000 Beziehungen zwischen diesen Knoten angelegt. Den größten Anteil dabei haben mit rund 276000 Knoten die Kommentare der Abgeordneten und Fraktionen. Die lange Laufzeit des Programms lässt sich erklären durch das Fehlen eines Batch-Prozesses für die Inserts in die Datenbank. Neomodel bietet zwar eine Möglichkeit für Batch-Inserts, aber es besteht dabei keine Möglichkeit gleichzeitig die Beziehungen mit anzulegen.
Ein weiteres Ergebnis und deren Visualisierung zeigt die Abbildung der Personen

```
{
  "identity": 1386,
  "labels": [
    "Commentary"
  ],
  "properties": {
"sentiment": 0.10182096561403509,
"dateString": "2018-04-25T13:00:00+00:00",
"applause": false,
"message": "Ich möchte, dass die Energiewende gelingt.
Dafür habe ich immer gearbeitet. Deshalb müssen wir
gemeinsam verstehen, dass der Ausbau der Netze – sowohl
der Ausbau der HGÜ, also der
Gleichstromübertragungsleitungen, wie auch der Ausbau der
380-kV-Leitungen – nach dem EnLAG vorankommen muss. Ich
würde mir sehr wünschen, dass Bündnis 90/Die Grünen und
alle anderen Fraktionen im Deutschen Bundestag die
Bundesregierung bei der Umsetzung des
Netzausbaubeschleunigungsgesetzes unterstützen würden,
das wir noch in diesem Jahr in den Bundestag einbringen."
  }
}
```


Abbildung 5.6: Ausgabe eines Kommentars

und des Sendens eines Kommentars in Richtung einer anderen Person oder Fraktion. Wir haben uns für die Darstellung dieses Graphen in die Dokumentation entschieden, weil der Fokus der Arbeit auf den Kommentare und deren Sentiments liegt. Die Graphen, die in Neo4j angezeigt werden können, sind schnell sehr komplex und können nur in kleinen Mengen (default: 300 Nodes) dargestellt werden.

5.6 Ausblick

(Autoren: Jennifer Vormann | Florian Thom)

Künftig wäre es eines unserer Ziele, die Laufzeit der Anwendung zu verbessern. Hierzu könnte generell noch einmal über den Code gegangen und dieser optimiert werden. So könnte die Runtime gegebenenfalls verbessert werden, um schnelles Schreiben in die Graphdatenbank zu ermöglichen. Zusammenfassend können folgende Punkte für eine Weiterentwicklung analysiert und gegebenenfalls umgesetzt werden:
- Arbeiten mit Batch
- Libraries | Programmiersprache variieren

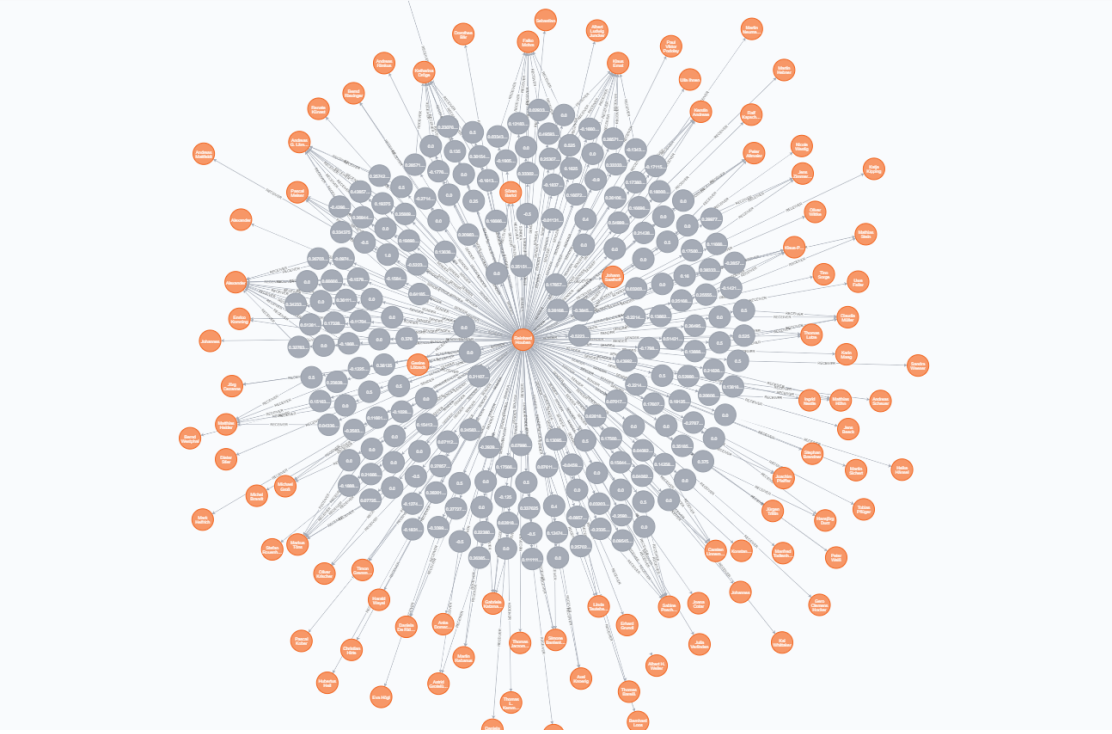

Abbildung 5.7: Neo4J Graph -> Person - Sender - Commentary - Receiver - Person

Ebenso wäre es ein Anliegen, die weiteren Wahlperioden zu ergänzen. Bei Optimierungen des Codes würden wir aller Wahrscheinlichkeit nach auf die Verwendung von neomodel verzichten und auf die Standard Implementierung mit Neo4j Python Driver ausweichen. Bei der Erweiterung um andere Wahlperioden wäre zu überdenken, die Wahlperiode als eigenen Node zu erstellen. Dazu müsste das Datenbankschema entsprechend angepasst werden.
Gerade im Entwurf wurden gewisse Optimierungen vorgestellt, bei denen aus theoretischer Sicht teilweise gute Verbesserungsmöglichkeiten gesehen wurden. Anfänglich wurde über einen praktischen Test oder Vergleich der Schemata spekuliert. Dieser Test blieb leider aufgrund von gewissen Schwierigkeiten in der Implementierung aus und könnte perspektivisch noch durchgeführt werden. Verbesserungsmöglichkeiten sind aus unserer Perspektive bezüglich des finalen Entwurfes vorhanden. Diese bestehen besonders in der Einbindung eines TLTs. Dies wurde mit dem ersten Entwurf aus theoretischer Sicht erläutert.

5.7 Lernziele

(Autorin: Jennifer Vormann)

Die Lernziele dieses Projektes und des damit verbundenen Kurses waren sehr vielfältig. In den Vorlesungen wurde Grundlagenwissen in den folgenden Bereichen vermittelt:
- Informationssysteme allgemein & Dateninspektion
- Crawling Textanalyse & NoSQL
- Sentiment Analyse & Graphdatenbanken

Aufgrund der Art der Durchführung der Übungen konnte das Format der Plenarsitzungen, inklusive deren Rollen, erlernt werden. Das Einsetzen realer Daten und deren Verarbeitung gehört zu den erreichten Lernzielen, ebenso wie das Extrahieren der relevanten Informationen aus den gegebenen Daten. Die Daten mussten betrachtet und bewertet werden anhand ihrer Relevanz für das Ziel der gesamten Projektaufgabe. Anschließend wurde das Aggregieren, Verknüpfen und Darstellen der Daten erlernt.
Zur Erstellung des Datenbankschemas wurden Kenntnisse zu Klassendiagrammen in Notation der UML vertieft und angewandt. Für die Zwischen- und Abschlusspräsentation der Vorgehensweise, konnte unser Wissen über das Schreiben eines Proof of Concept und das ansprechende Erstellen von Präsentationen gefestigt werden. Wir haben den Umgang mit Neo4j und deren deklarative Abfragesprache Cypher neu erlernt. Bei der Verarbeitung der Daten die von Gruppe 3 geliefert wurden und beim Weiterleiten unserer Ergebnisse an Gruppe 7 haben wir erfolgreich Schnittstellen von verschiedenen Datenbanken hergestellt. Unser Vorgehen und die Wahl der Technologien für die gesamte Umsetzung unserer Teilaufgabe lagen komplett in unserer Hand. Daher mussten wir abwägen, recherchieren und Vor- und Nachteile evaluieren. Die Kommunikation und Kollaboration war nicht nur in unserem Team wichtig sondern auch teamübergreifend. Auch das Verfassen einer kursumfassenden Dokumentation benötigte ein merklich höheres Maß an Abstimmung, Organisation und Kommunikation. Eigenständiges Arbeiten und Praktiken des Zeitmanagements konnten festigen werden.

Kapitel 6

Analyse der Interaktion zwischen Fraktionen

Gruppenmitglieder: Atanas Denkov, Mayk Akifovski, Roman Krajewski

6.1 Einleitung

Für die Analyse der Interaktionen zwischen Fraktionen sollen sowohl Interaktionen ausgewertet werden, die explizit von einer Fraktion an eine andere Fraktion gerichtet sind, als auch Interaktionen zwischen Fraktionsmitgliedern auf Fraktionsebene betrachtet werden. Ziel der Analyse soll sein, alle protokollierten Interaktionen auszuwerten und in Form eines Graphen, der die Sentiments zwischen Fraktionen darstellt abzulegen. Fraktionen spielen eine entscheidende Rolle im Deutschen Bundestag. Sie dürfen u.a. Gesetzentwürfe bzw. Änderungsanträge von solchen einbringen, kleine und große Anfragen im Bundestag stellen, und Sondersitzungen des Bundestags erzwingen. Das bedeutet, dass Fraktionen und deren gegenseitigen Aussprachen "die politische Willensbildung maßgeblich mitbestimmen" [33]. Dabei ist zu beachten, dass Fraktionsmitglieder ein freies Mandat bekleiden und damit zu abweichenden Meinungen gegenüber ihrer Fraktion befugt sind. [34] Daraus ergibt sich für die Aufgabenstellung das Problem, inwiefern Interaktionen zwischen Fraktionsmitgliedern in einen Graphen, der die Sentiments zwischen Fraktionen zeigen soll einfließen können. Spiegelt die Meinung eines Fraktionsmitglieds genau die Meinung der zugehörigen Fraktion wider? Können Interaktionen, die im Protokoll als von einer Fraktion ausgehend markiert sind auf alle ihre Mitglieder übertragen werden?

Wie mit dieser Frage umgegangen wurde wird im weiteren Verlauf erläutert. Weiterhin wird beschrieben, wie die gefundene Lösung technisch umgesetzt wurde, indem auf Anforderungen, genutzte Technologien und Endergebnisse eingegangen wird. Abschließend werden Optimierungsmöglichkeiten der Applikation dargestellt.

6.2 Grundlagen

6.2.1 Fraktionen

Eine Fraktion bezeichnet eine freiwillige Vereinigung von Abgeordneten des Bundestages, die ohne Konkurrenz ihre politische Ziele gemeinsam verfolgen und i.d.R derselben Partei angehören. Es ist darauf hinzuweisen, dass Fraktionen nicht mit Parteien verwechselt werden sollen. Erstere dienen dem öffentlichen Interesse und letztere dem Privatinteresse. Das heißt, dass Fraktionen aus öffentlichen Mitteln finanziert werden. Weiterhin folgt daraus, dass Fraktionsmitglieder gesetzmäßig an Aufträge und Weisungen ihrer Partei nicht gebunden und nur ihrem Gewissen unterworfen sind, auch wenn das meist einheitliche Abstimmungsverhalten eine Bindung suggeriert. [34] Es können also Aussagen im Parlament nicht eins zu eins von Fraktionsmitglied auf Fraktion übertragen werden.

6.2.2 Verwendete Daten

Aus der von von Gruppe 2 gestellte Datenbank werden in diesem Abschnitt Interaktionen mit Sentimentwert, sowie deren Sender und Empfänger verwendet. Sender und Empfänger können dabei entweder eine Fraktion oder eine Person sein. Die Fraktionszugehörigkeiten der Personen wird dabei auch der Datenbank entnommen. Obwohl eine Fraktionszugehörigkeitshistorie vorhanden ist, wird der Einfachheit halber immer die aktuelle Fraktion einer Person verwendet, da Fraktionswechsel innerhalb einer Legislaturperiode sehr selten und damit vernachlässigbar sind. Weiterhin wird zur Analyse die eine Rednerliste jeder Sitzung zu Rate gezogen. Aus diesen lässt sich eine Anzahl an Personen ableiten, die in dieser Sitzung für eine Fraktion das Wort ergriffen haben.

6.2.3 Gewichteter Mittelwert

Um gewichtete Sentiments auszuwerten wurde der gewichtete Mittelwert verwendet. Dieser ist bei n Werten $x_{1\ldots n}$ und Gewichten $w_{1\ldots n}$ wie folgt definiert:

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i \cdot x_i}{\sum_{i=1}^{n} w_i}$$

6.3 Anforderungsanalyse und Konzept

6.3.1 Aufgabenstellung

Aufgabe war es, unter Verwendung der von Gruppe 2 und Gruppe 3 zur Verfügung gestellten Daten einen Graphen aufzubauen, aus dem die aggregierten Sentiments zwischen Fraktionen ablesbar sind. Dieser Graph sollte in einer Graphdatenbank abgelegt werden und über eine Netwerkschnittstelle abrufbar sein.

6.3.2 Anforderungsanalyse

Zur Anforderungsanalyse wurden zunächst die vorliegenden Daten und deren Schnittstelle betrachtet. Die Daten sind in einer MongoDB abgelegt, auf die unsere Arbeitsgruppe einen Lesezugriff hat. Daraus ergibt sich die erste Anforderung unserer Anwendung: Daten aus einer MongoDB müssen gelesen werden können.

Anschließend wurde die Schnittstelle zur Anwendung der Gruppe 7 festgelegt. Da diese eine Analyse des Graphen vornehmen möchten wurde und damit möglichst individuellen Zugriff auf die Datenbank braucht, wurde auf das festlegen einer REST-Schnittstelle verzichtet. Stattdessen wurde sich darauf geeinigt, dass Gruppe 7 Zugriff auf die Graphdatenbank erhalten wird. Daraus ergibt sich also die Anforderung an unsere Applikation einen Zugriff auf die Datenbank zur Verfügung zu stellen. Im weiteren Verlauf des Projekts stellte sich heraus, dass die Sentiments der Interaktionen gewichtet abgelegt werden müssen, was die Abfrage mittels Queries etwas erschwert. Hierbei wurde sich darauf geeinigt, dass wir eine Query zur Verfügung stellen, die Interaktionssentiments aggregiert, welche als Grundlage für Queries an unsere Datenbank dient.

Eine Weitere Anforderung an unsere Anwendung besteht darin, die Abgebildeten Interaktionen nach Datum, Sitzung, Fraktion und Applaus filtern zu können. Daraus ergibt sich die Anforderung, dass diese als Felder in der Datenbank vorhanden sein müssen, sodass sie per Query gefiltert werden können. Das Filtern nach Applaus ergab sich hierbei im Laufe des Projekts, da überproportional viele Interaktionen, die nur aus Applaus bestehen, auftreten. Deren Sentiments können das Gesamtergebnis verzerren und sollen deshalb filterbar sein.

Eine weitere Anforderung ergibt sich aus dem Datenbestand. Da sowohl Interaktionen zwischen Personen auf deren Fraktion übertragen werden sollen, als auch Interaktionen zwischen Fraktionen an sich in den Graphen aufgenommen werden sollen, müssen deren Sentiments in irgendeiner Form gewichtet werden. Eine Anforderung an unsere Applikation ist also, mit Sentiments so umzugehen, dass für die dazugehörige Fraktion unterschiedlich stark ins Gewicht fallen, je nachdem wie viele Personen an der Interaktion beteiligt waren.

Um die Datenbank kontinuierlich zu füllen, sobald neue Sitzungen stattfinden,

sollte ein Mechanismus eingebaut werden, der für das aktualisieren der Datenbank sorgt, sobald neue Daten vorliegen. Dazu wurde sich darauf geeinigt, einen REST-Endpoint zu implementieren, mit dem die in der Datenpipeline vor unserer liegenden Applikation eine Aktualisierung auslösen kann.

6.3.3 Konzept

Um die Ständige Erreichbarkeit und Aktualisierung zu ermöglichen, wurde unsere Anwendung als Serverapplikation konzeptioniert. Diese sollte aus zwei unabhängigen Teilen bestehen, also der Graphdatenbank und der Aggregationsapplikation. Für die Graphdatenbank wurde Neo4J ausgewählt.

Aggregation

Um die Sentiments der Interaktionen zwischen Fraktionsmitgliedern proportional zu Sentiments zwischen Fraktionen zu aggregieren wurden diese zunächst gewichtet. Dabei wurden zwei Formeln angewandt, die im folgenden erklärt werden sollen.

Für die Interaktion zwischen Fraktionsmitgliedern a und b wird den Sentiments das Gewicht

$$w = \frac{1}{N_{F_a}} \cdot \frac{1}{N_{F_b}}$$

zugeordnet. Dabei stehen N_{F_a} und N_{F_b} jeweils für die Fraktionsgröße der Personen a und b. Diese Formel wird angewandt, damit ein Sentiment zwischen zwei Fraktionsmitgliedern nicht direkt auf ihre Fraktion übertragen wird. Beispielsweise könnten sich zwei Fraktionen gut verstehen, obwohl zwei ihrer Mitglieder sich überhaupt nicht verstehen. Mit oben beschriebener Gewichtung wird in diesem Fall einem Sentiment zwischen den Beiden nur so viel Gewicht gegeben, wie sie selbst Anteil der Fraktion sind.

Für Interaktionen zwischen einem Fraktionsmitglied a und einer Fraktion k wurde ebenfalls eine Formel ausgearbeitet. Da in einer Sitzung meist nur eine Teilgruppe aller Fraktionsmitglieder anwesend ist, sollte ein Sentiment auch entsprechend der Anzahl anwesender Mitglieder gewertet werden. Daraus ergibt sich die Formel

$$w = \frac{1}{N_{F_a}} \cdot \frac{N_{a_k}}{N_k}$$

mit N_{F_a} als Anzahl der Fraktionsmitglieder der Fraktion von Person a, N_{a_k} als Anzahl der Anwesenden Mitglieder der Fraktion k und N_k als Gesamtanzahl der Mitglieder von Fraktion k.

Architektur

Die Serverapplikation wurde zunächst als Java Springboot Applikation konzeptioniert. Während der Implementierung sind damit leider technische Schwierigkeiten aufgetreten, weshalb das Konzept dann in einer Python Applikation umgesetzt wurde.

Für die oben beschriebenen Gewichtungen muss als erstes die Mitgliederanzahl jeder Fraktion ermittelt werden. Dazu wurde eine MongoDB Query entwickelt, die alle Fraktionsmitglieder, die bisher in den Protokollen der behandelten Legislaturperiode vorkommen, ihrer Fraktion zuordnet und zählt. Dieses Vorgehen sollte nach dem ursprünglichen Konzept in SpringBoot umgesetzt werden, wurde aber letztendlich der Grund für den Wechsel zu Python, da das ausführen eigener MongoDB Queries in Python deutlich einfacher zu realisieren ist.

Sobald die Mitgliederanzahl jeder Fraktion bestimmt ist sollen alle Sitzungen nacheinander Abgearbeitet werden und ihre Interaktionen samt dazugehöriger ausgerechneter Gewichte in der Graphdatenbank abegelegt werden.

Zur Abfrage aggregierter Sentimentwerte sollten Cypher Queries implementiert werden, die jeweils den gewichteten Mittelwert aller den Filterkriterien entsprechenden Interaktionssentiments berechnen. Da beim gewichteten Mittelwert durch die Summe aller Gewichte geteilt wird, ist es in diesem Fall egal, welche Anzahl an Interaktionen den Filterkriterien entsprechen. Das ist auch der Grund, warum die Gewichtungen gesondert in der Datenbank abgelegt werden.

Ein weiterer Teil der Applikation ist der Endpunkt zur Aktualisierung der Datenbank. Dieser sorgt dafür, dass die gesamte aktuelle Legislaturperiode neu eingelesen wird. Leider ist es notwendig, die gesamte Legislaturperiode neu einzulesen, da sich die Gewichte von Interaktionen vergangener Sitzungen in ändern können. Dies ist der Fall, wenn ein Fraktionsmitglied in dem neuen Sitzungsprotokoll vorkommt, das vorher in keinem Protokoll genannt war. Damit ändert sich die Gesamtzahl der Fraktionsmitglieder und somit auch die Gewichtungen.

6.4 Implementierung

Die wesentliche Eigenschaft des bereitgestellten Kommunikationsmodells ist, dass es Beziehungen beinhaltet, welche jeweils durch einen Sender, einen Empfänger und eine Gewichtung repräsentiert werden. Die natürliche Struktur der Daten spiegelt ganz offensichtlich die eines gerichteten Graphen wieder. Die Speicherung der Daten in einer Graph-Datenbank ermöglicht die direkte Erkennung ihrer immanenten Beziehungen, was bei tabellenartigen (SQL) oder dokumentenbasierten (NoSQL) Datenbanken ohne komplexe Anfragen deutlich schwieriger ist. Die Applikation entspricht also einer ETL[1]-Pipeline. Sie bezieht sich auf die Erfassung von Daten aus einem externen System und deren Überführung in einem domänespezifischen Format, das das Abfragen und die Analyse der Daten erleichtert.

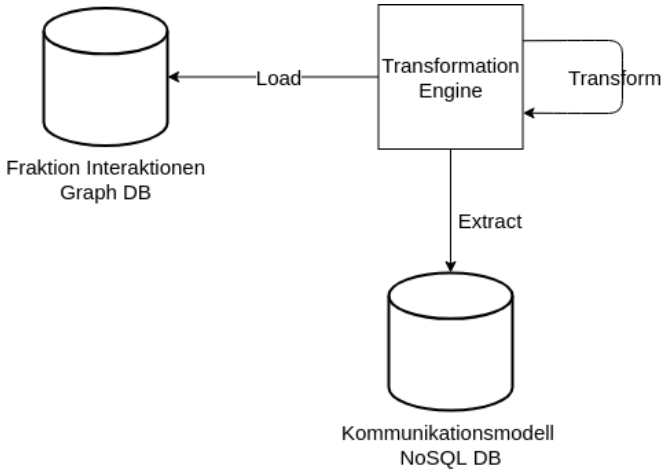

Abbildung 6.1: Allgemeiner ETL-Prozess zur Erfassung, Transformation und zum Laden von Daten

In diesem Fall handelt es sich um die Erfassung von Dokumenten, also Sitzungen des Bundestags, und deren Transformierung in einen gerichteten Graphen, dessen Knoten und Kanten jeweils die Fraktionen der gesamten Legislaturperiode und ihre gegenseitige Interaktionen sind. Abbildung 6.1 stellt eine grobe Architektur der ETL-Pipeline dar.
Die Anwendung besteht aus vier Komponenten - eine Komponente pro Phase in der Pipeline und eine HTTP-Schnittstelle zum Auslösen des gesamten Vorgangs, wenn neue Daten vorhanden sind.

[1]Extract, Transform, Load

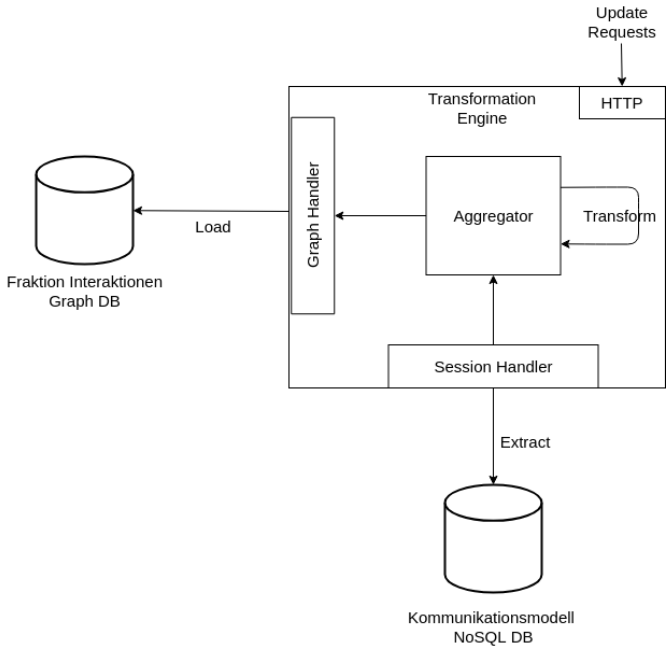

Abbildung 6.2: ETL-Architektur der Applikation

6.4.1 Erfassung

Die Kommunikation mit der externen Datenbank erfolgt über den sogenannten **Session Handler**, wie Abbildung 6.2 darstellt. Er stellt zwei komplexe Anfragen an die Datenbank, damit die Sitzungen in einer passenden Form in das Programm eingeladen werden können.
Die erste Anfrage gruppiert alle an einer Sitzung teilnehmende Abgeordneten nach der Fraktion der sie angehören. Auf diese Weise wird eine Hash-Tabelle erstellt, deren Schlüssel und Werte jeweils die Bezeichner der Fraktionen und die Liste mit den zugehörigen Bezeichnern der Abgeordneten ist:

$$\begin{aligned} F_1^{(i)} &\rightarrow [A_1^{(1)}, A_1^{(2)}, \ldots, A_1^{(N_1^{(i)})}] \\ F_2^{(i)} &\rightarrow [A_2^{(1)}, A_2^{(2)}, \ldots, A_2^{(N_2^{(i)})}] \\ &\vdots \\ F_m^{(i)} &\rightarrow [A_m^{(1)}, A_m^{(2)}, \ldots, A_m^{(N_m^{(i)})}]. \end{aligned} \tag{6.1}$$

$F_m^{(i)}$ ist die m-te Fraktion in der i-ten Sitzung und $A_m^{(1)}$ das erste Fraktionsmitglied der m-ten Fraktion. $N_m^{(i)}$ ist der Anzahl aller Fraktionsmitglieder der m-ten Fraktion in der i-ten Sitzung.

Die zweite Anfrage aggregiert die Interaktionen, je nachdem ob eine Person oder eine Fraktion die Interaktion begonnen bzw. empfangen hat. Auf diese Weise können die unterschiedliche Arten von Beziehungen separat behandelt werden, was die Lesbarkeit des Quellcodes wesentlich erhöht. Die Arten der Beziehungen wurden über Regular Expressions determiniert.
Das erfasste Datenmodell sah wie folgt aus:

```
class Faction:
    faction_id: str,
    name: str,
    members: list

class Session:
    session_id: str,
    legislative_period: int,
    start_date: str,
    end_date: str,
    pers_to_pers_comments: list,
    fac_to_pers_comments: list,
    pers_to_fac_comments: list,
    factions: List[Faction]
```

6.4.2 Transformierung

Mittels 6.1 kann eine Repräsentation für eine Fraktion in einer gesamten Legislaturperiode L abgeleitet werden.

$$F_m^L = \cup_{i=1}^{L} F_m^{(i)}. \tag{6.2}$$

6.2 vereinigt sukzessiv die eindeutigen Fraktionsmitglieder der m-ten Fraktion aus jeder Sitzung. Das bedeutet, dass N_m^L der Gesamtanzahl aller bisher in Legislaturperiode L anwesend gewesenen Fraktionsmitglieder entspricht. Somit sind alle Variablen zur Berechung der Gewichtungen vorhanden.
Der **Aggregator** (siehe Abbildung 6.1) iteriert über jede Interaktion, berechnet die Gewichtung und führt alle Eigenschaften der Interaktion zusammen mit der Gewichtung in einem Objekt über, das als eine Beziehung in einer Graph-Datenbank gespeichert werden kann.

6.4.3 Laden

Die Anzahl der Kommentare in einer Legislaturperiode beträgt i.d.R mehrere Hunderttausend (c.a 300000 für die 19. Legislaturperiode). Die Speicherkapazität eines

durchschnittlichen Rechners ist ungenügend, um alle Kommentare gleichzeitig in RAM aufzubewahren oder alle Kommentare auf einmal in der Graph-Datenbank zu speichern. Aus diesem Grund werden die Sitzungen hintereinander verarbeitet, wobei die Kommentare in einer Sitzung schubweise an die Graph-Datenbank weitergegeben werden.

6.4.4 Abfrage

Folgende Abfrage gibt eine Tabelle aus, die die aggregierten Sentiments zwischen allen Fraktionen angibt.

```
MATCH p=(sender:Faction)-[r:COMMENTED]->(receiver:Faction)
WITH sender, receiver, sum(r.weight) AS weightsum,
collect(r) AS rlist
UNWIND rlist as r
RETURN sender.name, receiver.name,
sum((r.weight/weightsum)*r.polarity)
```

Dazu werden zuerst alle Relationen mit einzigartigem Sender und Empfänger gruppiert und deren Gewichtssumme ausgerechnet. In diesem Schritt werden alle Relationen auch zu einer Liste zusammengefasst. Das ist notwendig, da in einer WITH expression alle Ausgaben die gleiche Anzahl an Werten für einen Grouping Key zurückgeben müssen. In diesem Fall wird jedem Sender-Empfängerpaar eine Gewichtssumme und eine Relationsliste zugeordnet, welche dann weiterverarbeitet wird. Im nächsten Schritt wird die Relationsliste wieder in ihre Elemente zerlegt um die sum() Akkumulationsfunktion benutzen zu können. Das Resultat der Query ist dann der gewichtete Sentimentmittelwert für jedes Sender-Empfänger-Paar.

6.5 Zusammenfassung und Ausblick

Ziel dieses Teilprojekts war es, eine Datentransformationsapplikation zur Analyse der Interaktionen zwischen Abgeordneten zu entwickeln. Es war notwendig, folgendes zu untersuchen und zu definieren:

- Modell der transformierten Enddaten und Datenbank zum Speichern der Ergebnisse für die weitere Analyse von Gruppe 7 und Gruppe 8.
- ein Backend Service, das in der Lage ist, die Sitzungsprotokolle abzufragen, sie in das entsprechende Modell zu transformieren und die Ergebnisse in der Datenbank zu speichern.
- eine Architektur, die auf dem HTW Server deployed ist und in der Lage ist auf Änderungen in der Datenpipeline zu reagieren und immer aktuelle und vollständige Daten zu liefern.

6.5.1 Ergebnisse

Im Laufe der Arbeit standen wir vor vielen Herausforderungen und mussten unseren ersten Aktionsplan ändern, damit er der gesamten Datenpipeline und den Anforderungen der Verbraucher entspricht - Gruppe 7. Zuerst wollten wir ein Java-Backend mit Spring Boot implementieren, mussten jedoch auf Python umsteigen, um Speicher- und Leistungsprobleme zu lösen. Wir glauben jedoch, dass wir alle unsere Ziele erfüllt und umgesetzt haben:

- eine Neo4J Datenbank steht zu verfügung und enthält die Interaktionen zwischen Fraktionen
- jeder Kommentar oder Reaktion zwischen Abgeordnete wird gewichtet.
- eine Python Backend-Applikation ist implementiert und ist als Docker Container auf den Uni-Server gehostet.
- der Service kann über eine HTTP-Rest Schnittstelle über neue Session-Ids benachrichtigt werden, ist in der Lage die neue Daten zu verarbeiten und in der Datenbank zu speichern.

6.5.2 Retrospektive und Verbesserungsmöglichkeiten

Wir bewerten die Ergebnisse und Erfolge unseres Projekts für den Proof-of-Concept-Meilenstein als erfolgreich. Obwohl wir es geschafft haben, unsere Ziele zu erreichen, gibt es Punkte, an denen wir unsere Arbeitsweise und die Funktionalität unserer Applikation besser gestalten konnten.

Als zukünftiges Ziel setzen wir uns den gleichzeitigen Code in unserer Applikation besser zu synchronisieren um viel Geschwindigkeit beim Lesen aus MongoDB und Schreiben in Neo4J zu gewinnen. Es wäre möglich unsere Service perfomanter und zuverlässiger zu gestalten, wenn wir folgende Werkzeuge ausprobieren und implementieren:

- Motor (Async Driver) für MongoDB - MongoDBs Queries werden als async Cursor definiert und die Daten werden erst geholt, wenn sie gebraucht werden. [35]
- Asynchronous I/O mit asyncio — eine high-level API in der standard Bibliothek von Python, die die Arbeit mit asynchrone Prozessen in Python vereinfacht. [36]
- concurrent.futures — Launching parallel tasks - eine high-level API um eine Menge von ähnliche Prozesse als einzelne Threads auszuführen. [37]

Außerdem haben wir schon nach vier Monate zusammenarbeit die SSchwächenünd Stärken unseres Team identifiziert. Das wäre uns sehr viel helfen, wenn wir unsere Arbeit in der Zukunft planen und die Afgaben zwischen uns verteilen.

Kapitel 7

Graphauswertung

Ermittlung der Stimmungsmacher im Bundestag
Gruppenmitglieder: Marie Bittiehn, Markus Christopher Glutting,
Miriam Lischke

7.1 Einleitung

Der folgende Teilabschnitt der Ausarbeitung beschäftigt sich mit dem sechsten Teilprojekt: die Graphauswertung. Die Projektgruppe, welche an dem Teilprojekt gearbeitet hat, besteht aus Markus Glutting, Miriam Lischke und Marie Bittiehn.

7.1.1 Hintergrund

Das Teilprojekt "Graphauswertung" baut auf den Ergebnissen der Teilprojekte "Interaktion zwischen Personen" (Gruppe 4) und "Interaktion zwischen Fraktionen" (Gruppe 5) auf. Genauer formuliert, besteht die Aufgabenstellung darin, die Graphen, welche von Gruppe 4 und 5 erstellt werden, auszuwerten und die Ergebnisse der Auswertung der nachfolgenden Gruppe 8 für ihre Benutzeroberfläche zur Verfügung zu stellen.

7.1.2 Problemstellung

Durch die Auswertung der Graphen sollen die sogenannten "Stimmungsmacher" im Bundestag ermittelt werden. Unter einem Stimmungsmacher ist im vorliegenden Kontext eine Person gemeint, welche viel mit vielen verschiedenen Personen redet und somit eine Stimmung verbreitet. Ob diese verbreitete Stimmung positiv oder negativ ist, ist dabei nicht entscheidend.

Neben der Ermittlung von Stimmungsmachern sollen ebenfalls simple mathematische Analysen auf den Graphen durchgeführt werden. Dadurch soll eine Gesamtbetrachtung der Sitzungen einer Wahlperiode ermöglicht werden, ebenso wie die Option die Sitzungen in Vergleich zueinander stellen zu können.

7.1.3 Zielsetzung

Um Stimmungsmacher zu ermitteln, soll der PageRank-Algorithmus (siehe Kapitel 7.2) verwendet werden. Dies bietet sich an, da Stimmungsmacher gleichbedeutend sind zu Personen, welche viele Nachrichten mit einem positiven und/oder einem negativen Sentiments empfangen bzw. versenden. Mithilfe des PageRank-Algorithmus werden eben diesen Personen bzw. Knoten im Graphen hohe Ränge vergeben, wodurch sie identifiziert werden können.

Für die mathematischen Analysen sollen Berechnungen auf den gesamten Graphen durchgeführt und statische Größen wie bspw. der Median oder ein Quartil der Sentiments berechnet werden. Die erstellten Analysen sollen der nachfolgenden Projektgruppe (Benutzeroberfläche) über Schnittstellen zur Verfügung gestellt werden.

Für die Bearbeitung des Teilprojekts wurden zu Projektbeginn folgende Lernziele identifiziert:

- Kennenlernen des PageRank-Algorithmus
- Kennenlernen von graphenorientierten Datenbanken, insbesondere Neo4j mit Cypher
- Wissensaufbau im Bereich Backend Webapplikationen

7.1.4 Prozess

Die Bearbeitung des Teilprojekts lässt sich in folgende vier Phasen gliedern:

- Phase 1 (Oktober 2020): Thematische Einarbeitung und Projektplanung
- Phase 2 (November - Dezember 2020): Implementierung der geplanten Features
- Phase 3 (Januar 2021): Anpassungen in Rücksprache mit der nachfolgenden Projektgruppe (Benutzeroberfläche)
- Phase 4 (Februar 2021): Dokumentation

Die Umsetzung des Projekts erfolgte arbeitsteilig durch alle Mitglieder des Projektteams. Darüber hinaus bestehen folgende Verantwortlichkeiten:

Tabelle 7.1: Gruppe 7 (Graphauswertung) - Aufgabenverteilung

Aufgabe	Gruppenmitglieder
Server-Administration	Markus Glutting
Implementierung: PageRank	Marie Bittiehn
Implementierung: Stammdaten	Alle
Implementierung: Statistische Analysen	Miriam Lischke
Organisation und Projekt Setup	Markus Glutting
Dokumentation	Alle

Die Entwicklung erfolgte in wöchentlichen Zyklen. Zu Beginn eines jeden Zyklus fand die wöchentliche Plenarsitzung im Rahmen der Lehrveranstaltung statt. Im Anschluss daran erfolgte eine gruppeninterne Absprache, in der die zuletzt abgeschlossenen Aufgaben besprochen und die nächsten zu erledigenden Aufgaben identifiziert wurden. Diese Aufgaben wurden mithilfe von GitHub-Issues dokumentiert und der verantwortlichen Person zugewiesen. Bis zur nächsten Plenarsitzung erfolgte die eigenverantwortliche Bearbeitung der Aufgaben.

7.2 Grundlagen

In diesem Kapitel werden die notwendigen Kenntnisse für das Verständnis der nachfolgenden Kapitel vermittelt. Zuerst wird der PageRank-Algorithmus erläutert mit Fokus auf eine effizientere Berechnung der Ränge mithilfe von Matrizenmultiplikationen. Nachfolgend wird auf die wichtigsten verwendeten Entwicklungsframeworks und -Tools eingegangen.

7.2.1 PageRank

Der PageRank-Algorithmus ist ein Algorithmus zur Gewichtung von Knoten innerhalb eines Netzwerks anhand ihrer Anzahl an eingehenden Beziehungen. Dabei geht in die Berechnung des PageRanks eines Knoten, neben seiner Anzahl an eingehenden Beziehungen, ebenso der PageRank der auf ihn verweisenden Knoten mit ein.

Aufgrund der daraus entstehenden direkten Abhängigkeit des PageRank eines Knoten von den PageRanks der auf ihn verweisenden Knoten, ist vor allem bei großen Netzwerken ein genaue Berechnung aller PageRanks nicht immer in absehbarer Zeit möglich. Aus diesem Grund werden die Werte für die PageRanks bei der Berechnung meist iterativ angenähert. Dabei wird allen Knoten ein einheitlicher Startwert als PageRank vergeben. Meist wird als Startwert der Wert $1/N$ verwendet, wobei N die Anzahl aller Knoten des Netzwerks ist. Danach wird der PageRank für alle Knoten mehrfach berechnet und so die Werte iterativ angenähert.

Eine Möglichkeit dieser iterativen Annäherung ist ein mehrfaches Iterieren über das gesamte Netzwerk und die rekursive Berechnung des PageRanks für jeden Knoten. Da im vorliegenden Anwendungskontext mit mehreren hundert Knoten und zig Tausend von Beziehungen zwischen den Knoten gerechnet werden kann, wäre das Iterieren über alle Knoten und Beziehungen nicht sehr effizient. Stattdessen können allerdings Matrizenmultiplikationen bzw. eine Eigenvektorberechnung für die PageRank-Berechnung verwendet werden.

Das Netzwerk, welches aus mathematischer Sicht als gerichteter Graph betrachtet werden kann, kann als eine quadratische, stochastische Matrix abgebildet werden [38]. Stochastisch bedeutet in diesem Kontext, dass die Summen der Spalten der Matrix alle 1 betragen [38]. Wenn im Graphen von einem Knoten j eine Beziehung zum Knoten i besteht, dann wird in der Matrix an der Stelle ij der Wert $1/d^+(j)$ eingetragen, wobei $d^+(j)$ für den Ausgangsgrad von j steht [38]. Wenn keine Beziehung besteht, wird an der Stelle ij eine 0 eingetragen [38]. Für den Fall, dass ein Knoten j gar keine ausgehenden Beziehungen besitzt, wird in seiner gesamten Spalte j in der Matrix der Wert $1/N$ eingetragen [38]. Dadurch wird verhindert, dass der PageRank in Sackgassen-Knoten gewissermaßen “versickert”.

Wurde der Graph nun als Matrix abgebildet, so kann der PageRank iterativ mithilfe von Matrixmultiplikationen berechnet werden. Dabei wird der PageRank als Vektor der Länge N dargestellt, meist befüllt mit dem Startwert $1/N$ für alle Knoten. Statt einer gesamten Iteration über das Netzwerk und den rekursiven Berechnungen neuer PageRanks für alle Knoten, muss nun nur eine simple Multiplikation der Matrix mit dem Vektor durchgeführt werden. Das Ergebnis der Multiplikation ist wieder ein Vektor und dieser enthält die neuen PageRanks aller Knoten. Nach mehrfacher Multiplikationen der Matrix mit dem PageRank-Vektoren ändern sich die Werte innerhalb des Vektor nicht mehr. Es wurde iterativ ein dominanter Eigenvektor der Matrix berechnet, welcher gleichzeitig den endgültigen PageRanks der Knoten des Netzwerks entspricht [38].

Auf gleiche Art und Weise kann der sogenannte "Reverse PageRank" berechnet werden. Bei dem Reverse PageRank handelt es sich um den PageRank berechnet auf dem Reverse-Graphen des Originalgraphen [39]. Dies bedeutet, dass der Reverse PageRank nicht basierend auf den eingehenden Beziehungen der Knoten berechnet wird, sondern auf den ausgehenden Beziehungen.

Um im ersten Schritt den Reverse-Graphen des Originalgraphen zu erhalten, werden die Richtungen der Beziehungen zwischen den Knoten des Originalgraphen umgedreht [39]. Danach wird der so erzeugte Reverse-Graph ebenfalls als Matrix abgebildet und die PageRank-Werte der Knoten werden mithilfe von Matrizenmultiplikationen berechnet. Die dadurch ermittelten Werte für den Reverse PageRank ähneln den Ergebnissen des klassischen PageRank dahingehend, dass die einzelnen Werte gleich sind, aber anders verteilt. Dies ist dadurch zu erklären, dass bei der Berechnung des Reverse PageRank zwar die Richtungen der Beziehungen umgedreht werden, jedoch keine Beziehungen entfernt oder hinzugefügt werden. Aus diesem Grund sind die Werte gleich, aber durch das Umdrehen der Beziehungen sind sie anders verteilt.

7.2.2 Entwicklungsframeworks und -Tools

Vor der Wahl eines Entwicklungsframeworks galt es zunächst zu entscheiden, welche Programmiersprache verwendet wird. Innerhalb der Gruppe war die Python-Bibliothek `numpy` bekannt, welche zur performanten Berechnung von Matrizen geeignet ist [40]. Daher fiel die Entscheidung auf die Programmiersprache Python. Zur Bereitstellung von HTTP-Schnittstellen für die Benutzeroberfläche wurde das Microframework "Flask" verwendet. Flask zeichnet sich dahingehend aus, dass es einen sehr schlanken Kern hat, der nach Bedarf erweitert werden kann [41]. Dadurch unterscheidet es sich stark von anderen Frameworks wie bspw. Django, welches stattdessen eher darauf abzielt, ein vollständiges Framework für Webanwendungen zu sein [42].

Der Zugriff auf die Neo4j Datenbaken erfolgt mithilfe des Neo4j Python-Treibers,

welcher im Python-Paket `neo4j` bereitgestellt wird [43]. Es wurde kein Object Graph Mapper verwendet, da eines der Lernziele darin bestand, die Abfragesprache Cypher kennenzulernen.

7.3 Anforderungsanalyse und Konzept

Die allgemeinen Anforderungen ergeben sich aus der Aufgabenstellung:

- Ermittlung von Stimmungsmachern im Bundestag
- Mathematische Analysen bezüglich der Sentiments im Bundestag

Zur Identifikation detaillierter Anforderungen fand im frühen Projektverlauf eine Absprache mit Gruppe 8 statt, welche die Benutzeroberfläche entwickelt. Im Rahmen dieses Meetings wurde festgelegt, dass folgende Funktionalitäten notwendig sind:

- Abfrage von Stammdaten:
  - Sitzungen
  - Fraktionen
  - Abgeordnete
- Abfrage des Graphen für Fraktionen und Abgeordnete. Die Übertragung des Graphen erfolgt durch gleichzeitige Übertragung der beteiligten Knoten (Abgeordnete bzw. Fraktionen) und der Nachrichten, die zwischen diesen verschickt wurden. Dabei werden Nachrichten zwischen zwei beteiligten Knoten aggregiert, um die übertragene Datenmenge zu reduzieren.
- Mathematische Analysen bezüglich des Sentiments im Bundestag. Als Ziel wurde die Darstellung von Boxplots angegeben. Daher wurde festgelegt, dass Minimum, unteres Quartil, Median, oberes Quartil und Maximum berechnet werden. Zum Vergleich der Sentiments innerhalb verschiedener Sitzungen soll die Möglichkeit der Filterung nach Sitzung vorgesehen werden.

7.3.1 Architektur

Der zentrale Bestandteil des Teilprojekts Graphauswertung ist die Backend Applikation. Hier werden die Daten, die zuvor über die Datenbankmanager aus den Neo4j Datenbanken von Gruppe 4 und 5 gezogen werden, zusammengeführt und weiter verarbeitet. Die Daten werden unter anderem zu statistischen Kennzahlen aufbereitet und mittels des PageRank und Reverse PageRank analysiert. Anschließend werden diese Daten über HTTP-Schnittstellen für die Benutzeroberfläche zur Verfügung gestellt.

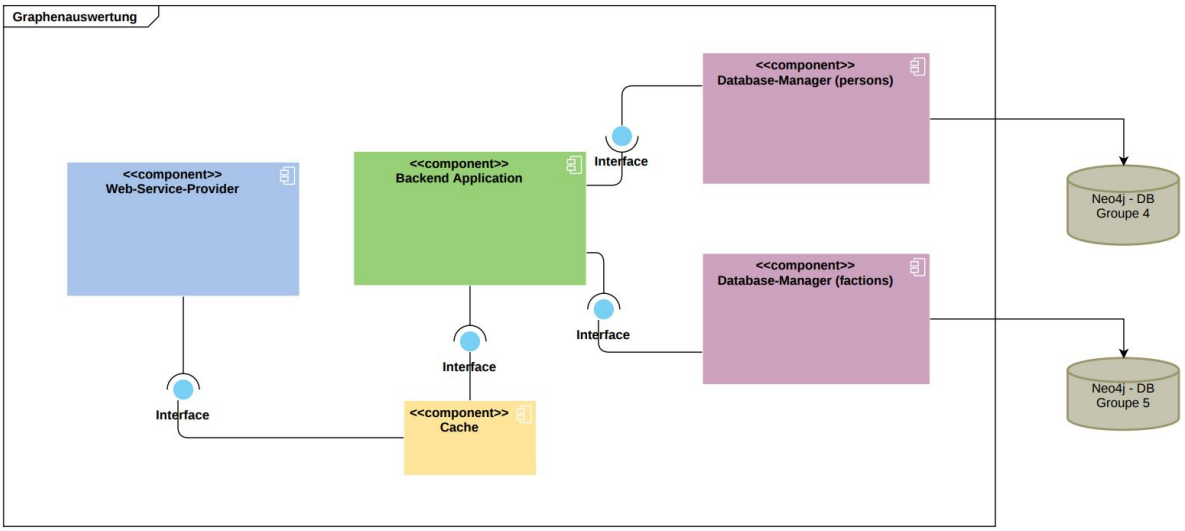

Abbildung 7.1: Graphauswertung - Komponentendiagramm,
Quelle: Eigene Darstellung, Tool: [44]

7.3.2 Schnittstellen

Im Folgenden werden die HTTP-Schnittstellen zu dem angrenzenden Teilprojekt (Benutzeroberfläche) beschrieben und an einigen Beispielen erklärt, welche Daten dabei bereitgestellt werden. Als vorletztes Teilprojekt in der Datenkette wird als Backend für die Benutzeroberfläche agiert. Dabei werden die Daten aus den Neo4j Datenbanken herausgezogen, aufbereitet und über HTTP-Endpunkte dem Frontend in verschiedener Form zur Verfügung gestellt. Die gesamte Dokumentation und Beispielausgaben der Endpunkte sind in der README im GitHub-Repository *Sentiments-of-Bundestag/Graphenauswertung* [45] zu finden.

Die HTTP Schnittstellen wurden mithilfe des Microframework Python Flask [41]

umgesetzt. Alle verfügbaren Endpunkte sind über GET erreichbar. Einige Endpunkte bieten mittels beigefügter Parameter Filtermöglichkeiten an.

Tabelle 7.2: Abfrageparameter

Name	**Beschreibung**	**Erwarteter Wert**
filter	Filtert die Nachrichten nach dem gegebenen Sentiment Typen.	POSITIVE, NEGATIVE
session	Filtert die Nachrichten nach der gegebenen Parlamentssitzungen.	sessionId aus `/sessions`
person	Filtert die Nachrichten nach den gegebenen Abgeordneten.	speakerId aus `/persons`
excludeApplause	Filtert alle Nachrichten aus, die mit `applause` markiert sind.	true
reverse	Wendet für die PageRank-Analyse den Reverse PageRank an.	true

Sessions:
Der Endpunkt **`/sessions`** gibt eine Liste aller verfügbaren Parlamentssitzungen zurück.

```
[
  {
   "sessionId": 138,
   "legislativePeriod": 19,
   "endDateTime": "2019-12-20T15:39:00+00:00",
   "startDateTime": "2019-12-20T09:00:00+00:00"
  },
  ...
]
```

Factions:
Der Endpunkt **`/factions`** gibt eine Liste aller Fraktionen zurück. Die Werte in `sessionIds` stehen für die Parlamentssitzungen, in denen Nachrichten mit anderen Fraktionen oder Abgeordneten ausgetauscht worden sind.

Factions graph:
Der Endpunkt **`/factions/graph`** gibt den Fraktionsgraphen basierend auf den Graphen der Datenbank von Gruppe 5 zurück. Die Nachrichten sind bereits aggregiert, also steht jede Verbindung zwischen Sender und Empfänger für alle Nachrichten in die jeweilige Richtung. Der Wert `count` ist die Anzahl aller für den aggregierten Eintrag berücksichtigten Nachrichten. Der Wert `sentiment` ist der Durchschnitt aller Sentiments aus den - für diesen aggregierten Eintrag berücksichtigten - Nachrichten. Der Durchschnitt ist zusätzlich durch die Fraktionsgröße gewichtet. Diese Schnittstelle bietet die Filteroptionen: person, session, excludeApplause, POSITIVE und NEGATIVE.


```
{
  "factions": [
   {
    "name": "CDU/CSU",
    "size": 246,
    "factionId": "F000"
   },
   {
    "name": "SPD",
    "size": 152,
    "factionId": "F001"
   }
  ],
  "messages": [
   {
    "recipient": "F001",
    "sender": "F000",
    "sentiment": -1.399999976158142,
    "count": 10,
    "sessionIds": [46, 100]
   },
   {
    "recipient": "F000",
    "sender": "F001",
    "sentiment": 0.1,
    "count": 2,
    "sessionIds": [46, 100]
   }
  ]
}
```

Persons:
Der Endpunkt **`/persons`** gibt eine Liste aller Abgeordneten zurück. Diese Schnittstelle bietet die Filteroptionen: person, session, POSITIVE und NEGATIVE.

```
[
    {
    "faction": "CDU/CSU",
    "factionId": "F000",
    "name": "Frank Heinrich",
    "role": "Platzhalter",
    "speakerId": "MDB-24aa7763-e95d-4d1d-834c-de3cae2406d7",
    "sessionIds": [ 46, 100 ]
},
...
]
```

Messages:
Der Endpunkt **`/persons/messages`** gibt eine Liste aller Nachrichten zurück. Die Nachrichten sind nicht aggregiert. Diese Schnittstelle bietet die Filteroptionen: session, person, excludeApplause, POSITIVE und NEGATIV.

Persons graph:
Der Endpunkt **`/persons/graph`** gibt den Abgeordnetengraphen, basierend auf den Graphen der Datenbank von Gruppe 4 zurück. Die Nachrichten sind bereits aggregiert, also steht jede Verbindung zwischen Sender und Empfänger für alle Nachrichten in die jeweilige Richtung. Der Wert `count` ist die Anzahl aller für den aggregierten Eintrag berücksichtigten Nachrichten. Der Wert `sentiment` ist der Mittelwert aller Sentiments aus den - für diesen aggregierten Eintrag berücksichtigten - Nachrichten. Diese Schnittstelle bietet die Filteroptionen: person, session, excludeApplause, POSITIVE und NEGATIVE.

Factions ranked:
Der Endpunkt **`/factions/ranked`** gibt eine Liste aller Fraktionen, geordnet nach dem PageRank zurück. Diese Schnittstelle bietet die Filteroptionen: reverse, session, excludeApplause, POSITIVE und NEGATIVE.

Persons ranked:
Der Endpunkt **`/persons/ranked`** gibt eine Liste aller Abgeordneten, geordnet nach dem PageRank zurück. Diese Schnittstelle bietet die Filteroptionen: reverse, session, excludeApplause, POSITIVE und NEGATIVE.

```
[
  {
   "faction": "DIE LINKE",
   "factionId": "F003",
   "name": "Caren Lay",
   "rank": 0.029715244473085708,
   "role": "Platzhalter",
   "sessionIds": [ 46, 100 ],
   "speakerId": "MDB-c3f825cc-9b63-4241-85f9-df425f0c6486"
  },
  {
   "faction": "CDU/CSU",
   "factionId": "F000",
   "name": "Wolfgang Schaeuble",
   "rank": 0.029176551464191278,
   "role": "Platzhalter",
   "sessionIds": [ 46   ],
   "speakerId": "MDB-fd366231-9c25-416b-8604-e934d956e177"
  }
]
```


Factions proportions:
Der Endpunkt **`/factions/proportions`** gibt eine Liste aller Fraktionen mit dem jeweiligem Redeanteilen als Prozentwert zurück. Diese Schnittstelle bietet die Filteroptionen: session und excludeApplause.

Factions sentiment key figures:
Der Endpunkt **`factions/sentiment/keyfigures`** gibt den Sentimentwert aller Fraktionen verrechnet in statistische Kennzahlen zurück. Diese Schnittstelle bietet die Filteroptionen: session und excludeApplause.


```
{
 "highest_sentiment": 16.04166666790843,
 "lowest_sentiment": 0.5,
 "sentiment_lower_quartile": 1.0,
 "sentiment_median": 2.5903602056205273,
 "sentiment_upper_quartile": 5.3949188850820065
}
```


Persons sentiment key figures:
Der Endpunkt **`persons/sentiment/keyfigures`** gibt den Sentimentwert aller Abgeordneten verrechnet in statistische Kennzahlen zurück. Diese Schnittstelle bietet die Filteroptionen: session und excludeApplause.

7.4 Implementierung

In diesem Kapitel werden Einzelheiten zur Umsetzung der entwickelten Applikation beschrieben. Des Weiteren erfolgt eine Darstellung der notwendigen Schritte zur Bereitstellung der Anwendung auf dem HTW-Server.

7.4.1 Umsetzung

Die Backend Anwendung wurde als Python Flask Applikation entwickelt. Der gesamte Quellcode ist im GitHub-Repository *Sentiments-of-Bundestag/Graphenauswertung* [46] verfügbar.

Die Applikation wird durch Ausführen der Datei `app.py` gestartet. Diese stellt zunächst die Verbindung mit den beiden Neo4j Datenbanken her. Zur Kommunikation mit den beiden Datenbanken existiert jeweils eine Klasse, die den Zugriff auf die jeweilige Datenbank verwaltet und Methoden zur Abfrage der Inhalte der Datenbank bereitstellt. Die Zugangsdaten für die Datenbanken werden als Umgebungsvariablen bereitgestellt. Diese werden beim Start der Applikation aus der Datei `.env` eingelesen. Aus Sicherheitsgründen, ist diese Datei nicht Teil der Quellcodeverwaltung. Einige komplexere Prozeduren wie bspw. die Berechnug des PageRank wurden ausgelagert und werden von den Datenbankklassen importiert. Die Definition der Schnittstellen erfolgt in Form von Methoden, die mit dem Pfad annotiert sind, unter dem die jeweilige Schnittstelle erreichbar ist. Aufgrund der großen Datenmenge und der damit verbundenen hohen Laufzeit werden alle Anfragen für 12 Stunden gecached. Dazu wird die Bibliothek Flask-Caching mit dem Cache-Typ SimpleCache verwendet, welcher einem In-Memory Python-Dictionary entspricht [47]. Die Implementierung der Schnittstellenmethoden besteht in der Regel aus einem Aufruf einer Methode der Datenbankklassen, wobei das Ergebnis zur Übertragung in der HTTP-Response in das JSON-Format umgewandelt wird.

Die Methoden der Datenbankklassen sind alle nach dem gleichen Muster aufgebaut. Die nachfolgende Methode, welche die verfügbaren Fraktionen abfragt, kann als Beispiel dienen.

```
1  def get_factions(self,
2                   sentiment_type="NEUTRAL",
3                   session_id=None):
4
5      where = self.get_where_clause(sentiment_type, session_id)
6      with self.driver.session() as session:
7          query = \
8              "MATCH (sender:{0})-[r:{1}]-(recipient:{0}) " \
9              "{2} " \
10             "with sender, " \
11             "collect(distinct r.sessionId) as sesList, " \
```

```
12              "collect(r) as rlist unwind rlist as r " \
13              "RETURN DISTINCT sender.name as name, " \
14              "sender.size as size, " \
15              "sender.factionId as factionId, " \
16              "sesList as sessionIds" \
17              .format(NODE_FACTION, REL_COMMENTED, where)
18
19      factions = session.run(query)
20      return factions.data()
```

Listing 7.1: Zugriff auf Neo4j DB: get_factions

Die Methoden sind stets so aufgebaut, dass zunächst eine Datenbanksitzung geöffnet wird. Zur Datenbankabfrage wird eine Cypher Query konstruiert. Diese hängt von den Parametern der HTTP-Anfrage ab, welche in der `WHERE` Klausel der Query abgebildet werden. Da viele Methoden die gleichen Parameter unterstützen, ist die Konstruktion der `WHERE` Klausel in eine eigene Methode ausgelagert. Die Query wird anschließend an die jeweilige Neo4j Datenbank geschickt. Das Resultat wird in einigen Fällen noch weiter verarbeitet und dann als Ergebnis zurückgegeben. Mit dem Abschluss der Methode wird auch die Datenbanksitzung beendet.

Eine Herausforderung bei der Anbindung der Datenbanken bestand darin, dass beide zwar auf Neo4j basieren, jedoch wurde die Beziehung zwischen zwei Knoten (Abgeordnete bzw. Fraktionen) unterschiedlich abgebildet. Im Graphen der Abgeordneten wurde eine Nachricht zwischen zwei Knoten als zusätzlicher Knoten abgebildet, während im Graphen der Fraktionen eine Nachricht als Beziehung implementiert wurde. Die Arbeit mit der Datenbank der Abgeordneten wurde uns dadurch erleichtert, dass die verantwortliche Gruppe uns beispielhafte Cypher Queries zur Verfügung gestellt hat.

7.4.2 Bereitstellung

Zur Bereitstellung der Anwendung auf einem Server wurde ein Docker Image erstellt, mithilfe dessen die Anwendung gestartet werden kann. Dabei wurde wie in [48] beschrieben vorgegangen. Das Docker Image basiert auf dem Standard Docker Image für Python und wird gebaut, indem zunächst die Bibliotheken, von denen die Anwendung abhängig ist, installiert werden. Anschließend werden der Quellcode und die Konfigurationsdateien in das Docker Image kopiert. Zum Start der Anwendung wird ein Shell Skript aufgerufen, welches den uWSGI Applikationsserver startet. HTTP-Anfragen auf dem Standard Port 80 werden mithilfe von nginx an den uWSGI Server weitergeleitet. Um den Start der Anwendung ohne Angabe des Port Mappings zu gestalten, wurde außerdem eine `docker-compose` Datei erstellt.

Zur Bereitstellung auf dem HTW-Server (`http://infosys6.f4.htw-berlin.`

`de/`) wurde Port 80 in der Firewall geöffnet und git und Docker installiert. Der Start der neuesten Version der Anwendung auf dem Server erfolgt durch folgende Schritte:

1. Zugriff auf den Server via `ssh`
2. Herunterladen der neuesten Version des Quellcodes mithilfe von `git clone` bzw. `git pull`
3. Erstellung des neuen Docker Images mithilfe von `docker-compose build`
4. Start der Applikation mithilfe von `docker-compose up`

Die verfügbaren Schnittstellen wurden in der `README` des GitHub-Repository im Detail beschrieben und so der nachfolgenden Gruppe (Benutzeroberfläche) zugänglich gemacht. Die Benutzeroberfläche verwendet die Schnittstellen seither als Datengrundlage für sämtliche Diagramme auf der Website.

7.5 Zusammenfassung und Ausblick

In diesem abschließenden Kapitel erfolgt eine Zusammenfassung der Ergebnisse. Weiterhin werden die Lernziele bewertet und ein Ausblick über Verbesserungsmöglichkeiten gegeben.

7.5.1 Zusammenfassung

Im Rahmen des Teilprojekts Graphauswertung wurde eine Backend Applikation entwickelt, welche der Benutzeroberfläche die benötigten Daten zur Verfügung stellt. Diese Daten umfassen Stammdaten, PageRank sowie statistische Kennzahlen und werden als HTTP-Schnittstellen bereitgestellt.

7.5.2 Lernziele

Die gesetzten Lernziele wurden vollumfänglich erreicht. Durch die thematische Einarbeitung konnten alle Gruppenmitglieder einen guten Einblick in den PageRank-Algorithmus gewinnen. Im Rahmen der Entwicklung wurden außerdem Kenntnisse im Bereich Neo4j / Cypher und Python Flask Backend Applikationen aufgebaut. Zuletzt erfolgte ein Wissensaufbau in der Arbeit mit Docker zur umgebungsunabhängigen Bereitstellung der Applikation.

7.5.3 Ausblick

Die geforderten Funktionalitäten wurden im Rahmen des Teilprojekts vollumfänglich umgesetzt. Dennoch bestehen einige Verbesserungsmöglichkeiten, die im Folgenden beschrieben werden.

Der gewählte Cache-Typ ist in seiner Funktionalität stark eingeschränkt. Durch die Verwendung eines Redis Cache [49] könnte diese verbessert werden, sodass bspw. eine automatische Aktualisierung ermöglicht wird und der Cache im Falle einer horizontalen Skalierung über mehrere Instanzen gelten kann. Diese Änderung wurde testweise auch umgesetzt, jedoch erst sehr spät im Projektverlauf. Um die Stabilität der Applikation während der Abschlusspräsentation nicht zu gefährden, wurde diese Änderung nicht mehr in die Anwendung integriert.

Die Bereitstellung neuer Versionen geht mithilfe weniger manueller Schritte vonstatten. Eine Automatisierung dieses Prozesses wäre durch die Verwendung von GitHub-Actions möglich. Allerdings setzt dies voraus, dass der verwendete GitHub-Runner den HTW-Server per `ssh` erreichen kann. Aus Sicherheitsgründen wurde dieser Schritt daher nicht umgesetzt.

Kapitel 8

Benutzeroberfläche

Gruppenmitglieder: Amanda Joelle Dzukou Kom, Steven Mi, Tilman Möller und Oliver Kütemeier

Die Gruppe 8 hat sich mit der Erstellung der Benutzeroberfläche beschäftigt. Die Implementation des Projektes wurde in enger Zusammenarbeit der Studenten umgesetzt, um eine gleichmäßige Verteilung der Kompetenzen zu sichern.

8.1 Einleitung

Die Benutzeroberfläche bildet die Schnittstelle zwischen den Ergebnissen der Analysen und den Benutzer*innen die mit dem System interagieren möchten. Ziel ist es, die komplexen Daten der Analysen, in für die Benutzer*in gut verständliche Visualisierungen und Testergebnisse aufzuteilen. Zusätzlich zu den Herausforderungen der Implementation haben wir uns mit den Schwierigkeiten des Designs und der Koordination bei Gruppenarbeiten auseinander gesetzt. Unterschiedliche Frontend Frameworks wurden betrachtet, Bibliotheken zu Visualisierung ausprobiert und Design Thinking Methoden angewendet um in der kurzen Zeit dieses Semesters eine solide und präsentable Webseite umzusetzen.

8.2 Auswahl des Frameworks

Bei der Implementation des Frontends bestand der erste Schritt in der Auswahl des Frameworks. Die drei betrachteten Kandidaten waren Vue.js, Angular und React.

Abbildung 8.1: Logos der zur Auswahl stehenden Frameworks

Nach erster Recherche wurde klar, dass alle Frameworks in der Lage sind moderne und stabile Webseiten zu erstellen. Sie unterstützen alle diverse Bibliotheken zur Visualisierung und bieten die Funktionalitäten die geplanten Inhalte umzusetzen. Da die drei Frameworks die technischen Anforderungen gleichmäßig erfüllen, konnten wir persönliche Kriterien aufstellen um eine Entscheidung zu treffen. Das erste Kriterium bestand in der aktuellen Nachfrage in der Wirtschaft. Dazu haben wir die Menge der Suchanfragen nach dem jeweiligen Skill in unterschiedlichen Bereichen untersucht. Angefangen haben wir mit den Jobbörsen und Business Social Networks.

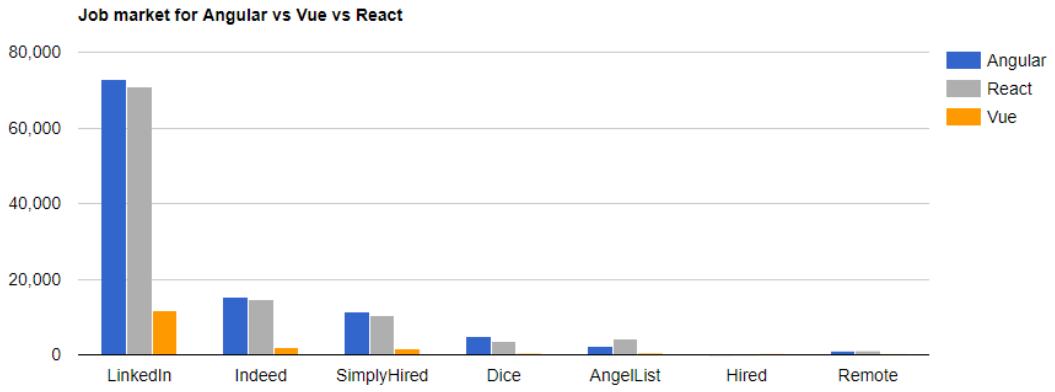

Abbildung 8.2: Vergleich der Suchanfragen in Jobbörsen und Business Social Networks

Hier ist klar, dass die Industrie mehr an den Frameworks React und Angular interessiert ist. Um die allgemeine Meinung besser einschätzen zu können haben wir die Google Trending Suchanfragen der verglichen.

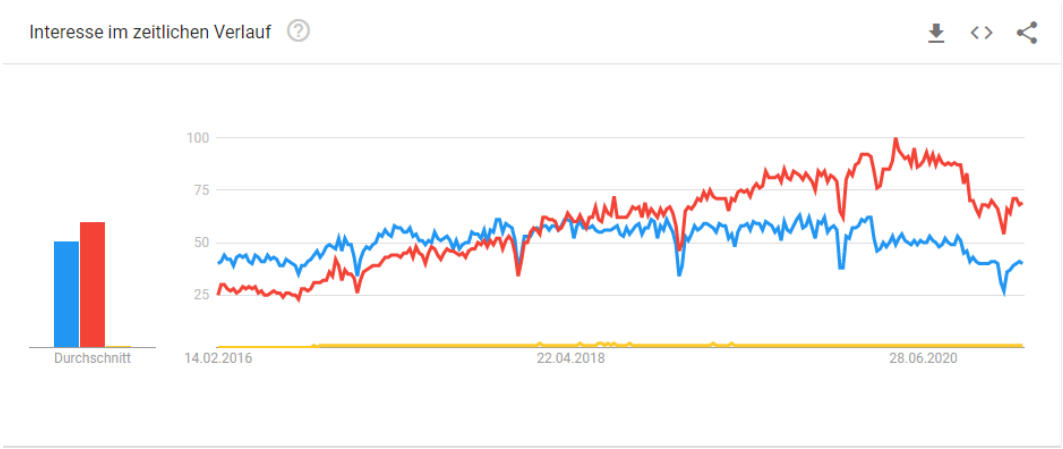

Abbildung 8.3: Vergleich der Google Trending Suchwörter

Auch hier stehen Angular und React weit vor den Anfragen von, sodass wir uns zwischen diesen beiden entscheiden mussten. Als letztes Kriterium haben wir die aktuellen Github Statistiken der Frameworks verglichen um einen Eindruck für die Community und dessen Aktivität zu erhalten.

Github Stats (stand 20.11.2020)	**Angular**	**Vue**	**React**
Watchers	3.2k	6.3k	6.8k
Stars	68k	175k	159k
Forks	18k	27.2k	31.6k
Contributors	1,275	381	1.522

Abbildung 8.4: Vergleich der Github Statistiken der Frameworks

Unter Berücksichtigung aller Kriterien haben wir uns für die Entwicklung der Website mit React entschieden.

8.3 Infrastruktur

Die Infrastruktur, welche unser Team zum verwalten des Quellcodes und zum bereitstellen der Webseite genutzt hat, war zum einen an die Absprachen der Studentengruppe und zum anderen an die Vorgaben der Hochschule gebunden. In Abstimmung mit allen Studenten wurde sich dazu entschieden, den Quellcode in Github zu verwalten. Die Bereitstellung erfolgte darüber hinaus über die Server der HTW Berlin, auf denen wir eine Virtuelle Maschine zur Verfügung gestellt bekommen haben.

8.3.1 Quellcode Verwaltung

Innerhalb der Github Organisation SSentiments-of-Bundestag"haben wir das Git Repository frontend_sentimentängelegt. Wir haben mithilfe der von Github eingeführten Issues eine Schnittstelle zum Kommunizieren von Problemen angelegt, die von den anderen Gruppen genutzt werden konnte. Über die Readme des Projekts haben wir alle wichtigen Befehle und Hilfestellungen zur Nutzung der Anwendung dokumentiert. Durch integrieren eines Linters in den Build-Prozess der Anwendung konnten wir als Gruppe feste Quellcode-Formatierungs-Regeln definieren und befolgen um Clean Code zu implementieren. Als zusätzliche Regelung zur Unterbindung von Problemen beim Programmieren haben wir in Branches gearbeitet und bei Änderungen am Quellcode Rücksprache im Team gehalten.

8.3.2 Bereitstellung

Bei der Bereitstellung der Anwendung haben wir die Serverkapazitäten der HTW Berlin genutzt. Uns wurde eine virtuelle Maschine zur Verfügung gestellt. Auf dieser mussten zunächst die Zugriffsregeln der Firewall angepasst werden um den Zugriff auf feste Ports, von außerhalb des HTW Netzwerks, zuzulassen. Außerdem mussten Git und Node.js installiert werden. Git wurde benötigt um den Quellcode aus dem Repository bei Änderungen herunterzuladen. Node.js wurde als Server zur Bereitstellung der React Anwendung benutzt.

8.4 Design Thinking

Beim Gestalten der Benutzeroberfläche haben wir uns an den sechs Phasen des von Tim Browns erfundenen Design Thinking Prozess orientiert.

Abbildung 8.5: Ablauf des Design Thinking Prozess

Dieser Prozess wird im Online Artikel "Was ist Design Thinking?"[50] gut beschrieben. In diesem Projekt standen wir zunächst vor dem Problem, dass wir die Daten und fest definierten Schnittstellen erst im letzten drittel des Semesters vorliegen hatten. Vorher haben wir uns mit der Auswahl der Graphen- und Diagrammtypen beschäftigt. Wir haben Prototypen entwickelt und diese in regelmäßigen Abständen mit der Studentengruppe geteilt und Rückmeldungen gesammelt, welch in den Design Prozess eingeflossen sind. Mit den Rückmeldungen der Gruppe konnten wir unseren Prototypen verbessern, Teile umgestalten Beschreibungen

anpassen oder Bereiche entfernen. Die erste Iteration bestand in der Vorbereitung der Planungspräsentation unserer Gruppe. Bis zu diesem Zeitpunkt haben wir Ideen gesammelt, wie die angekündigten Daten in Diagrammen und für Benutzer*innen verständlich aufbereitet werden können. In einem Prototypen haben wir vier unterschiedliche Diagramme vorbereitet.

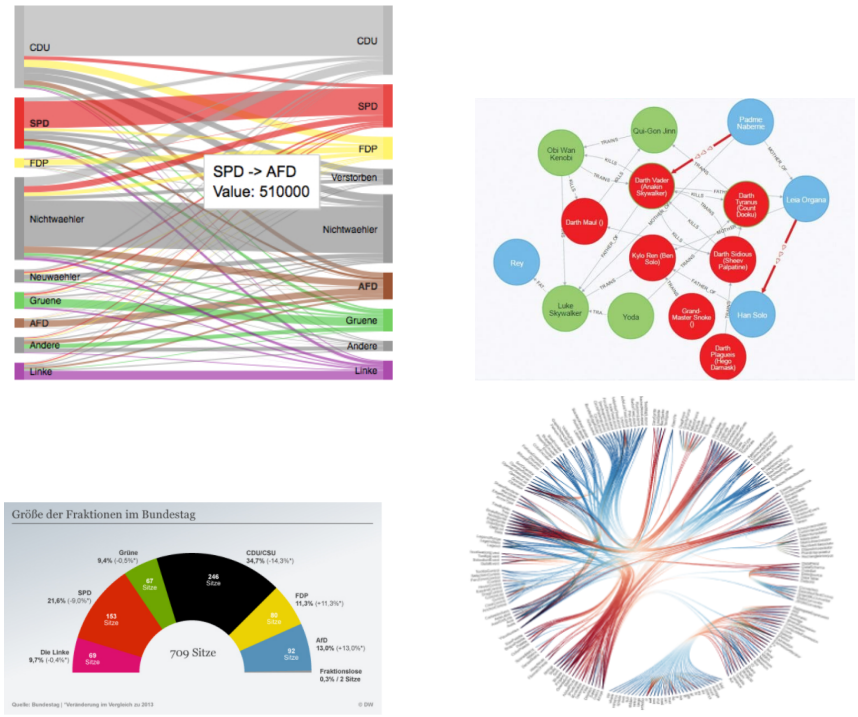

Abbildung 8.6: Vorbereitete Diagramme der ersten Iteration

Die Rückmeldungen und Diskussion am Ende der ersten Iteration, haben gezeigt, dass die direkte Visualisierung des Graphen zu unübersichtlich und nicht anschaulich ist. Ebenso ist das Sankey-Diagramm unübersichtlich und schwer zu interpretieren. Diese Diagramme sind nicht übernommen worden. Im Gegenteil dazu, wurden das Sehenendiagramm zur Visualisierung der Interaktionen und das halbe Kreisdiagramm zur Darstellung der Sitzeverteilung sehr positiv angenommen und deswegen weiter entwickelt.

In der zweiten Iteration wurden das Netzdiagramm und die Balkendiagramme integriert, die positives Feedback bekommen haben. Ebenfalls haben wir mit Box-Plots experimentiert, diese allerdings wegen der hohen Komplexität nicht weiter verfolgt.

Nach den ersten beiden Iterationen standen die Diagrammtypen fest und wir konnten mit den einbetten der Diagramme in die Oberfläche beginnen. Dazu wurden die Diagramme in ein Storyboard eingebunden und mit Texten beschrieben.

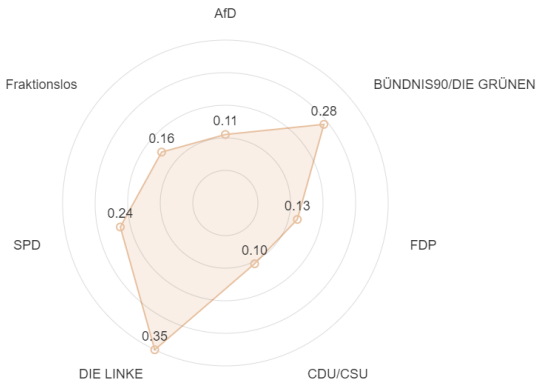

Abbildung 8.7: Eingeführtes Netzdiagramm

In den letzten Iterationen wurden Kleinigkeiten, wie die Texte und die Farben, abgestimmt um zum finalen Produkt zu gelangen.

8.5 Implementierung

8.5.1 Technologien

Die Benutzeroberfläche ist eine Single-Page-App, die hauptsächlich in React mit TypeScript geschrieben wurde. TypeScript typisiert JavaScript und ermöglicht die Vorteile einer typisierten Sprache. React Router DOM wurde verwendet, um die Navigation ohne eine Webseitenaktualisierung zu implementieren. Datenfluss und -verwaltung wurden mit Redux und Redux Sagas ermöglicht. Durch den Einsatz der vordefinierten UI-Elemente von Ant Design und Bootstrap konnten wir eine einheitliche Designsprache beibehalten. Letztlich wurden mit nivo die dynamischen und animierten Grafiken erstellt.

Abbildung 8.8: Eine Übersicht über alle verwendeten Technologien, Werkzeuge und Bibliotheken

React

React basiert auf dem sogenannten Komponentenmuster (engl. "component pattern"), in der jedes Element einer Webseite einer "stateless"Komponente entspricht. "Stateless"Komponenten sind wiederverwendbare Webseiten Elemente, welche keinen eigenen Status, Daten oder Inhalt besitzen. Sie dienen lediglich für die funktionale Transformationen von Eingabedaten.[51] Zudem kann jede Komponente

ebenfalls eine Tochterkomponente besitzen.

Die Daten und Komponenten werden von sogenannten Container verwaltet. Ein Container aggregiert die benötigten Komponenten und sorgt für den Datenfluss.

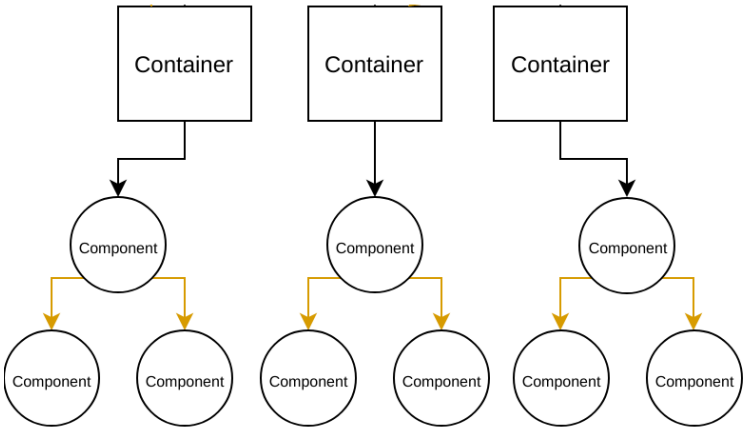

Abbildung 8.9: Die Visualisierung vom Datenfluss im Komponentenmuster. Eine Menge von Container besitzen Komponenten welche Daten erhalten und weiter an den Tochterkomponenten übergeben von [52]

Redux und Redux Sagas

Redux und Redux-Sagas wurde für den Datenfluss und -verwaltung verwendet.

Redux dient als ein globaler Speicherort für Informationen. Daher wird Redux oftmals auch als Redux Store bezeichnet. Container greifen auf den Redux Store zum um entweder Daten zu erhalten oder zu verändern. Dabei können Container sogenannte Actions ausführen, um Daten zu erhalten. Mithilfe von Reducer können Daten vorselektiert oder verändert werden.

Die Informationen im Redux Store müssen jedoch erst von einem Server angefordert werden. Dieser Vorgang ist jedoch asynchron und gibt keine Informationen über den Status des Prozesses zurück. Beispielsweise es ist nicht bekannt ob der Server die Daten verarbeitet oder ob der Prozess fehlgeschlagen ist. Der Anwender hingegen möchte wissen, ob Daten gerade verarbeitet werden. Um dies zu ermöglichen, wurde Redux-Sagas eingesetzt. Redux-Sagas ist eine Middleware, die sich zwischen der Benutzeroberfläche und dem Redux Store befindet. Wenn die Benutzerfläche eine Action ausführt, wird die Middleware Daten vom Server anfordern und dem Reducer weitergeben. Der Reducer wiederum bearbeitet die Daten und speichert sie im Redux Store.

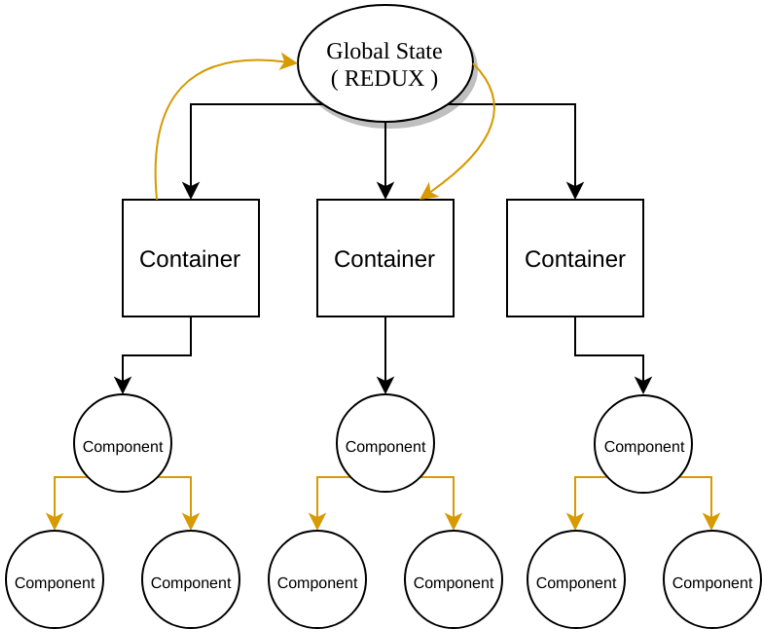

Abbildung 8.10: Die Visualisierung vom Datenfluss im Komponentenmuster mit Redux von [52]

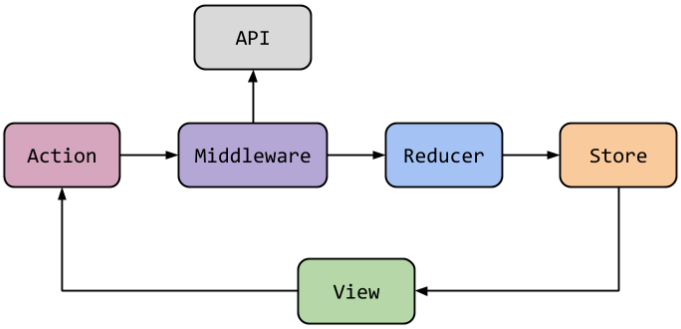

Abbildung 8.11: Die Visualisierung vom Datenfluss mit Redux und Redux-Sagas von [53]. Die View ist die Benutzeroberfläche, welche eine Action ausführt und den Prozess startet.

Ant Design, Bootstrap und nivo

Ant Design, Bootstrap und nivo sind sogenannte UI-Komponenten-Bibliotheken, Bibliotheken die vorgefertigte UI-Komponenten wie Buttons, Eingaben, Dialoge etc. bereitstellen. Sie dienen als Bausteine für Layouts und können modular verwendet werden. Jede Bibliothek hat ihre eigene Designsprache und damit ein cha-

rakteristisches Aussehen. Sie können jedoch in der Regel bis zu einem gewissen Grad angepasst werden.

Für Sentiment of Bundestag wurden Ant Design, Bootstrap und nivo verwendet. Ant Design wurde von der Firma Ant Design entworfen und stellt allgemeine UI-Komponenten wie Buttons, Tabellen und co. in der firmeninternen Designsprache zur Verfügung. Bootstrap stellt ähnliche Komponenten wie Ant Design zur Verfügung, allerdings in der Designsprache der Firma Twitter. Bei Nivo handelt es sich um ein Open-Source-Projekt, das vorgefertigte Diagramme anbietet, die mit d3.js und anderen React-Frameworks implementiert sind. Ihr Vorteil ist, dass die Diagramme animiert und für mobile Geräte optimiert sind.

8.6 Zusammenfassung und Ausblick

Durch die unterschiedlichen angewandten Methoden und die verwendeten Frameworks und Bibliotheken konnten wir viel Neues kennen lernen. Rückblickend konnten wir in diesem Semester eine solide und gut vorzeigbare Benutzeroberfläche für unser Informationssystem schaffen. Wir haben sehr viel positives Feedback erhalten und bei der Live-Demo und in den vorangegangenen Präsentationen hat die Webseite wie geplant funktioniert.

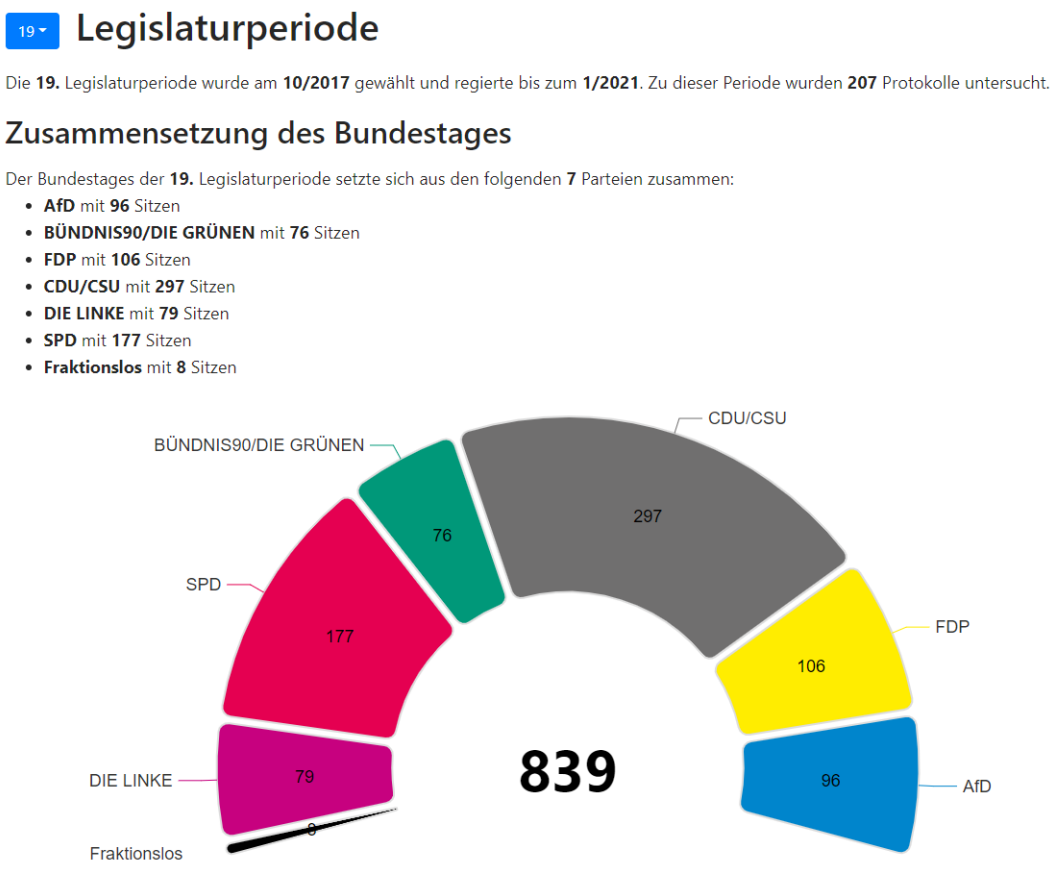

Abbildung 8.12: Screenshot aus der Benutzeroberfläche

Ein Aspekt den wir zu Beginn des Semesters unterschätzt haben war die Abhängigkeit von den vorangehenden Gruppen. Diese führte zu Zeitproblemen, Koordinationsaufwand und Problemen bei der Entwicklung. Diese Herausforderungen konnten wir nur im Team durch Zusammenarbeit und durch direkte Kommunikation mit den anderen Gruppen lösen. Alles in Allem sind wir mit dem Projekt sehr Zufrieden. Sowohl der Lernerfolg in diesem Semester als auch das dabei entstandenen Produkt sind überzeugend. Wir hoffen sehr, dass das Projekt in Zukunft

fortgeführt wird und andere Studierende den Design Thinking Prozess und die Entwicklung fortführen.

8.6.1 Erreichbarkeit

Die Webseite kann, für derzeit unbestimmte Zeit, unter http://infosys7.f4.htw-berlin.de/ erreicht werden. Sollte dieser Link nicht mehr erreichbar sein, bitte wenden Sie sich an Prof. Dr. rer. nat. Thomas Hoppe.

Literaturverzeichnis

[1] *Der raue Ton im Bundestag*. besucht am 10.01.2021, um 11Uhr. URL: https://www.zdf.de/nachrichten/heute-sendungen/videos/rauer-ton-im-bundestag-100.html#xtor=CS5-95.

[2] *Open Data*. besucht am 10.01.2021, um 11Uhr. URL: https://www.bundestag.de/services/opendata.

[3] Thomas Hoppe. „Informationssysteme: Übung“. In: *Informationssysteme*. 2020, S. 10–12. URL: https://moodle.htw-berlin.de/pluginfile.php/1015183/mod_resource/content/0/Planung%20%C3%9Cbung%20Teil%202.pdf.

[4] *Parlament*. besucht am 10.01.2021, um 11Uhr. URL: https://www.bundestag.de/parlament/plenum/abstimmung/liste.

[5] *Sitzungskalender*. besucht am 10.01.2021, um 11Uhr. URL: https://www.bundestag.de/parlament/plenum/sitzungskalender/sitzungskalender-196314.

[6] Thomas H. Cormen u. a. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009, S. 540–549. ISBN: 0262033844.

[7] Donald Knuth. „The Art of Computer Programming“. In: *The Art of Computer Programming: Second Edition*. Bd. 3. 2. Boston: Addison-Wesley Professional, 1998. Kap. Sorting and Searching, S. 73–168, 180–197, 392–478. ISBN: 978-0-201-89685-5.

[8] *Dokumentation CME*. besucht am 10.01.2021, um 11Uhr. URL: infosys2.f4.htw-berlin.de:9001/cme/doc/docs#/data/get_session_cme_data_session__session_id__get.

[9] *Spring Boot 2.4.2*. besucht am 10.01.2021, um 11Uhr. URL: https://spring.io/projects/spring-boot.

[10] *Crawler*. besucht am 10.01.2021, um 11Uhr. URL: https://github.com/Sentiments-of-Bundestag/Crawler.

[11] *HtmlUnit*. besucht am 10.01.2021, um 11Uhr. URL: https://htmlunit.sourceforge.io/.

[12] *flask Dokumentation*. URL: https://flask.palletsprojects.com/en/1.1.x/ (besucht am 23. 02. 2021).

[13] *flask Dokumentation Blueprints*. URL: https://flask.palletsprojects.com/en/1.1.x/blueprints/ (besucht am 23. 02. 2021).

[14] *waitress Webserver Dokumentation*. URL: https://docs.pylonsproject.org/projects/waitress/en/stable/ (besucht am 23. 02. 2021).

[15] *PyMongo Dokumentation*. URL: https://pymongo.readthedocs.io/en/stable/tutorial.html (besucht am 23. 02. 2021).

[16] *Interest Group on German Sentiment Analysis*. URL: https://sites.google.com/site/iggsahome/downloads (besucht am 23. 02. 2021).

[17] *SentimentWortschatz - Projekt Deutscher Wortschatz / Leipzig Corpora Collection*. URL: https://wortschatz.uni-leipzig.de/de/download (besucht am 23. 02. 2021).

[18] *Multi-Domain Sentiment Lexicon for German*. URL: https://sites.google.com/site/iggsahome/downloads/OPM.zip?attredirects=0 (besucht am 23. 02. 2021).

[19] *Polcla - A Polarity Classifier Incorporating Polarity Shifting for German*. URL: https://github.com/artificial-max/polcla (besucht am 23. 02. 2021).

[20] *github josefkr/morphcomp*. URL: https://github.com/josefkr/morphcomp (besucht am 23. 02. 2021).

[21] *Open DE WordNet Initiative*. URL: https://ikum.mediencampus.h-da.de/projekt/open-de-wordnet-initiative/ (besucht am 23. 02. 2021).

[22] *Wn Documentation*. URL: https://wn.readthedocs.io/en/latest/index.html (besucht am 23. 02. 2021).

[23] *spaCy Dokumentation v2.x*. URL: https://v2.spacy.io/models (besucht am 23. 02. 2021).

[24] Michael Wiegand, Maximilian Wolf und Josef Ruppenhofer. „Negation Modeling for German Polarity Classification“. In: *Language Technologies for the Challenges of the Digital Age*. Hrsg. von Georg Rehm und Thierry Declerck. Cham: Springer International Publishing, 2018, S. 95–111. ISBN: 978-3-319-73706-5.

[25] Melanie Siegel und Melpomeni Alexa. *Sentiment-Analyse deutschsprachiger Meinungsäußerungen: Grundlagen, Methoden und praktische Umsetzung*. Jan. 2020. ISBN: 978-3-658-29698-8. DOI: 10.1007/978-3-658-29699-5.

[26] *Sitzverteilung im 19. Deutschen Bundestag*. URL: https://www.bundestag.de/parlament/plenum/sitzverteilung_19wp. (aufgerufen am 30.01.2021).

[27] I. Robinson, J. Webber und E. Eifrem. *Graph Databases: New Opportunities for Connected Data*. O'Reilly Media, 2015. ISBN: 9781491930847. URL: https://books.google.de/books?id=jzvcCQAAQBAJ.

[28] *Understanding Neo4j's data on disk*. URL: https://neo4j.com/developer/kb/understanding-data-on-disk/. (aufgerufen am 30.01.2021).

[29] *The Secret Sauce of Neo4j: Modeling and Querying Graphs*. URL: https://neo4j.com/blog/secret-sauce-neo4j-modeling-graphconnect. (aufgerufen am 30.01.2021).

[30] *Splits relationship chains by type and direction*. URL: https://github.com/neo4j/neo4j/commit/366d30928d1b1590eba5daef92116ecc15aa36b1. (aufgerufen am 30.01.2021).

[31] *Fragestellungen*. URL: https://github.com/Sentiments-of-Bundestag/graphenauswertung/wiki/Fragestellungen. (aufgerufen am 30.01.2021).

[32] Noa Roy-Hubara u. a. „Modeling Graph Database Schema". In: *IT Professional* 19 (Nov. 2017), S. 34–43. DOI: 10.1109/MITP.2017.4241458.

[33] *Fraktion (BPP)*. besucht am 23.02.2021, um 17Uhr. URL: https://www.bpb.de/nachschlagen/lexika/handwoerterbuch-politisches-system/202022/fraktion.

[34] *Abgeordnete (BPP)*. besucht am 23.02.2021, um 17Uhr. URL: https://www.bpb.de/nachschlagen/lexika/handwoerterbuch-politisches-system/201755/abgeordneter.

[35] *Motor Docs*. besucht am 23.02.2021, um 17Uhr. URL: https://motor.readthedocs.io/en/stable/.

[36] *Async IO Docs*. besucht am 23.02.2021, um 17Uhr. URL: https://docs.python.org/3/library/asyncio.html.

[37] *Concurrent Futures (Python Docs)*. besucht am 23.02.2021, um 17Uhr. URL: https://docs.python.org/3/library/concurrent.futures.html.

[38] Taher Haveliwala. *Efficient Computation of PageRank*. Technical Report 1999-31. Stanford InfoLab, 1999. URL: http://ilpubs.stanford.edu:8090/386/ (besucht am 09. 02. 2021).

[39] Li-Tal Mashiach und Ziv Bar-Yossef. „Local approximation of pagerank and reverse pagerank". In: *Proceedings of the 17th ACM conference on Information and knowledge management*. Okt. 2008, S. 279–288. URL: http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi/2008/MSC/MSC-2008-09 (besucht am 18. 02. 2021).

[40] *numpy Website*. URL: https://numpy.org/ (besucht am 19. 02. 2021).

[41] *Flask Dokumentation*. URL: https://flask.palletsprojects.com/ (besucht am 19. 02. 2021).

[42] *Django Website*. URL: https://www.djangoproject.com/ (besucht am 19. 02. 2021).

[43] *Neo4j Bolt driver for Python*. URL: https://pypi.org/project/neo4j/ (besucht am 19. 02. 2021).

[44] *Visual Paradigm Tool*. URL: https://online.visual-paradigm.com/ (besucht am 20. 02. 2021).

[45] *Graphenauswertung*. URL: https://github.com/Sentiments-of-Bundestag/graphenauswertung#endpoints (besucht am 19. 02. 2021).

[46] *Graphenauswertung*. URL: https://github.com/Sentiments-of-Bundestag/graphenauswertung (besucht am 19. 02. 2021).

[47] *Flask-Caching Dokumentation*. URL: https://flask-caching.readthedocs.io/ (besucht am 19. 02. 2021).

[48] Alexey Smirnov. *Running Flask in production with Docker*. URL: https://smirnov-am.github.io/running-flask-in-production-with-docker/ (besucht am 19. 02. 2021).

[49] *Redis Website*. URL: https://redis.io/ (besucht am 20. 02. 2021).

[50] *Was ist Design Thinking?* besucht am 13.02.2021, um 12Uhr. URL: https://www.designerinaction.de/design-wissen/design-thinking/.

[51] *Components and Props*. besucht am 13.02.2021, um 12Uhr. URL: https://reactjs.org/docs/components-and-props.html.

[52] *React State Management Patterns*. besucht am 13.02.2021, um 12Uhr. URL: https://itnext.io/react-state-management-patterns-908325dbb8f3.

[53] *Handle side-effects with Redux-Saga*. besucht am 13.02.2021, um 12Uhr. URL: https://scalac.io/redux-saga-handle-side-effects-2/.

Glossar

API Application Programming Interface. 5, 32, 38–40

CLI Command Line Interface. 32

CME Communication Model Extractor. 30, 32, 37–40

HTW Hochschule für Technik und Wirtschaft. 40

IP Internet Protocol. 38

JSON JavaScript Object Notation. 30, 32, 34, 40

MDB Mitglied des Deutschen Bundestages. 32, 33, 36, 37

REST Representational State Transfer. 32, 37, 38

SaaS Software as a Service. 27

SoB Sentiments of Bundestag. 30

SSH Secure Shell. 37

XML Extensible Markup Language. 30, 32, 34, 36, 40