Technische Hochschule Lübeck

Fachbereich Elektrotechnik und Informatik

Interaktive Netzwerkanalysen am Beispiel von Twitter

im Studiengang
Master of Science Medieninformatik

zur Erlangung des akademischen Grades
Master of Science

Thema:	Interaktive Netzwerkanalysen am Beispiel von Twitter
Autor:	Ferenc Beutel
Version vom:	23. Januar 2020
1. Betreuer:	Prof. Dr. rer. nat. Dipl.-Inform. Nane Kratzke
2. Betreuer:	Prof. Dr.-Ing. Jens Ehlers

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

1 Einleitung

Big Data ist in den Medien. Das Themenfeld hat längst den Sprung von einem Teilgebiet der Informatik und Statisitik in das Innerste vieler Unternehmen jeglicher Größe und Branche geschafft. Es fallen heute so viele Daten an wie noch nie zu vor: Im Jahr 2017 fiel beispielsweise ein globales Traffic-Volumen von 8190 Exabytes an und das Wachstum in den nächsten Jahren wird als exponentiell eingeschätzt [Sta18] .
Da immer mehr Daten durch Nutzung des Internets anfallen und diese immer stärker genutzt werden, um das Verhalten der Nutzer zu verstehen und die Dienstleistungen gezielt und datengetrieben zu steuern, müssen die Prozesse, die diese Daten erzeugen, verarbeiten und so aufbereiten, dass ein Mensch sie verstehen kann, angepasst werden. Dafür ist es unbedingt notwendig, eine skalierbare, erweiterbare und robuste Architektur für die für die Datenverarbeitung verantwortlichen Softwarekomponenten zu wählen.
Durch die verstärkte Abhängigkeit und Nutzung von diesen Daten werden sie immer wertvoller für die Unternehmen, die diese Daten besitzen. Dies könnte dazu führen, dass Unternehmen versuchen, diese Daten geheim zu halten, um einen Vorteil gegenüber Wettbewerbern zu erhalten. Diese Einstellung war vor einigen Jahren noch stark verbreitet. Inzwischen hat es aber ein Umdenken in der Branche gegeben: So genannte Public APIs, also öffentliche Zugänge zu den Daten, die das Unternehmen besitzt, werden von vielen Unternehmen angeboten. Der Grund dafür ist, dass viele Unternehmen nicht die Ressourcen haben, um ihre Daten in dem Umfang zu nutzen, wie es für tausende Individuen möglich ist. Somit wird ein Teil der Produktentwicklung aus dem Unternehmen in die Öffentlichkeit ausgelagert, und das meist kostenlos.[Die17] Weiterhin ist es für Unternehmen auch möglich, eine gute API, die Zugang zu wertvollen Daten bietet, zu monetarisieren. Ein Beispiel dafür ist die Google Maps API [Clo].
Auch Twitter bietet die Möglichkeit, diverse Datensätze über eine Public API abzurufen. Als Social Network haben die Daten von Twitter eine stark vernetzte Struktur, sie bilden ein Netzwerk. Dieses Netzwerk kann genutzt werden, um diverse Fragestellungen über die Nutzer von Twitter zu beantworten. Vernetzte Daten ermöglichen es, komplexe Zusammenhänge zu beantworten und weitreichende Muster-Erkennungen durchzuführen. Allerdings stellen sich auch besondere Herausforderungen bei der Arbeit mit stark vernetzten Daten. Wie kann man die Daten beispielsweise intuitiv und verständlich darstellen? Oder wie lassen sich diese Daten effizient speichern, sodass die Analysen, die durchgeführt werden sollen, einfach realisierbar sind?
Für diese Arbeit wurde der deutschsprachige Twitter-Traffic für einen Monat mitgeschnitten. Daraufhin wurde 1 Prozent des aufgezeichneten Traffics ausgewählt und als Grundlage für diese Arbeit zur Verfügung gestellt.

1.1 Motivation

Anhand von Twitter lassen sich viele Fragestellungen, die die Gesellschaft und das soziale Miteinander betreffen, beantworten. [Edi+10] Für die Analyse dieser Daten gibt es bereits einige Services im Internet, jedoch kaum wissenschaftliche Beschreibungen, wie die Architektur so einer Plattform aussehen müsste. Häufig sind Informationen zu bestimmten Teilaspekten zu finden, das große Ganze fehlt aber.
Die Analyse von Daten in sozialen Medien stellt im Generellen ein sehr ergiebiges Forschungsfeld da. Durch die bereits 100 Jahre zurückreichende wissenschaftliche Geschichte der klassischen Analyse sozialer Strukturen, gestartet von unter Anderem William Lloyd Warner, [Fre04] und die Aktualität und Neuheit sozialer Medien wie Twitter führt gerade die Kombination dieser beiden Felder zu vielen neuen Erkenntnissen, die durch die ausgereiften wissenschaftlichen Grundlagen gestützt werden. Da die Forschung häufig den Menschen und seine Interaktionsmuster in den Mittelpunkt stellt, versprechen die Ergebnisse von hoher Relevanz für die Menschheit zu sein.

1.2 Ziel dieser Arbeit

Das Ziel dieser Arbeit ist es, Erkenntnisse über die Verarbeitung, Aufbereitung, Visualisierung und das Durchsuchen von stark vernetzten Massendaten zu gewinnen. Da der Arbeit ein Datensatz mit über 4,5 Millionen Rohdateneinträgen zu Grunde liegt, treten bereits viele Herausforderungen auf, die spezifisch für die Arbeit mit Big Data sind. Insbesondere soll die Frage beantwortet werden, wie eine geeignete Softwarearchitektur für ein Analysesystem aussieht, dass vom Import über die Verarbeitung bis hin zur Darstellung der Daten alle notwendigen Funktionalitäten für die Analyse eines sozialen Netzwerks am Beispiel des vorliegenden Datensatzes bietet. Hierbei soll insbesondere auf die stark vernetzte Natur des Datensatzes eingegangen werden. Ein thematischer Schwerpunkt soll auf der Erarbeitung einer interaktiven und intuitiven Darstellungsform für den vorliegenden Datensatz liegen.
Es wird im Rahmen dieser Arbeit ein Prototyp entwickelt, der einen kompletten technischen Durchstich analog zur entwickelten Architektur bieten soll. Es soll möglich sein, mit Hilfe dieses Prototyps die vorhandenen Daten zu importieren, sie in eine Form zu bringen, die die vorhandenen Netzwerkaspekte betont, und diese anschließend zu visualisieren. Dabei soll es insbesondere möglich sein, die Daten nach verschiedenen Aspekten zu durchsuchen und das Netzwerk interaktiv erkunden zu können.

1.3 Gliederung

Nach dieser Einleitung werden im Kapitel 2 zunächst einige Grundlagen zu verschiedenen Themen, die für diese Arbeit von Relevanz sind, aufgearbeitet. In Kapitel 3 wird daraufhin auf die theoretischen Grundlagen des Kernthemas dieser Arbeit, die Visua-

lisierung von stark vernetzten Massendaten, eingegangen.
Daraufhin wird in Kapitel 4 eine Anforderungsanalyse durchgeführt, die sowohl fachliche als auch technische Aspekte beinhaltet. Diese werden aus der Literaturarbeit sowie der Zielsetzung dieser Arbeit abgeleitet. Kapitel 5 beschreibt anschließend eine Systemarchitektur, die sowohl das Backend als auch das Frontend inklusive einer Deployment-Strategie beinhaltet.
In Kapitel 6 wird die technische Realisierung der Anforderungen beschrieben. Hierbei werden die Anforderungen auf die Architektur projeziert, um daraus die möglichen Lösungsvorschläge zu den verschiedenen Anforderungen zu gewinnen und diese anschließend zu diskutieren. Kapitel 7 beinhaltet anschließend eine Beschreibung des implementierten Prototypen.
In Kapitel 8 findet die Validierung der Ergebnisse statt. Kapitel 9 fasst die Ergebnisse zusammen, bewertet diese und gibt einen Ausblick.

2 Grundlagen

Dieses Kapitel zeigt die Grundlagen auf, auf die die restliche Arbeit aufbaut.

2.1 Twitter

Wenn nicht spezifisch zitiert wurden die folgenden Informationen [Twib] entnommen. Bei Twitter handelt es sich um ein soziales Netzwerk, dass es den Usern ermöglicht, Kurznachrichten zu bestimmten Themen mit anderen Usern zu teilen und auf solche Nachrichten anderer User zu reagieren [Kwa+10].
Hierbei gibt es eine Reihe von Interaktionsmöglichkeiten eines Users. Er kann beispielsweise twittern, was nichts anderes bedeutet, als einen Tweet, also eine Kurznachricht, zu erstellen, und diese zu veröffentlichen. Diese Kurznachricht kann neben dem eigentlichen Text auch andere Medieninhalte beinhalten, beispielsweise Hyperlinks, Bilder oder Videos. Tweets sind dabei auf 280 Zeichen begrenzt [Kur17].
Weiterhin können Tweets Verweise auf andere Inhalte innerhalb von Twitter beinhalten: Wird ein Wort mit dem Symbol @ gepräfixed, erzeugt dies eine Referenz auf einen User mit genau diesem Namen. Gibt es diesen User, erstellt Twitter aus der Referenz einen Link zum Profil des verlinkten Users. Diese Technik nennt Twitter mentions, also Erwähnungen. Der erwähnte User wird auch über die Erwähnung benachrichtigt und kann daher leicht darauf reagieren [Pin18].
Tweets können noch eine andere Art von Referenz beinhalten. Wird ein Wort in einem Tweet mit dem Symbol # gepräfixt, führt dies zu einer thematischen Verlinkung. Diese Verlinkung nennt Twitter Hashtag. Sämtliche Tweets, die ein Hashtag beinhalten, beinhalten eine Verlinkung auf eine Themenseite, die dynamisch für das entsprechende Hashtag angelegt wird. Hashtags bieten somit eine thematische Gruppierung von Tweets [Pin18].
Eine weitere Interaktion, die User durchführen können, ist das so genannte Retweeten. Dabei handelt es sich um eine Aktion, die ein User mit dem Tweet eines anderen Users durchführen kann. Durch einen Retweet klont der User den entsprechenden Tweet und veröffentlicht diesen unter seinem Profil mit einer Referenz zum ursprünglichen Autor. Es handelt sich also um eine Art von Teilen von Inhalten, denn somit erhält der entsprechende Tweet eine größere Reichweite, da ihn prinzipiell mehr Menschen sehen können, da er in mehr Twitter-Profilen zu finden ist [Pin18].
Eine Spezialisierung eines Retweets ist das Quoten, also Zitieren von anderen Tweets. Hierbei handelt es sich um einen Retweet, den der retweetende User noch um ein Kommentar erweitern kann.
Eine weitere mögliche Interaktion ist das Liken von Tweets. Hiermit signalisiert ein User, dass ihm der Content eines anderen Users gefällt, indem er einen Tweet des Users mit einem Like versieht. Dies ist beispielsweise mit dem gefällt-mir Knopf von

Facebook vergleichbar.
Schlussendlich können Usern anderen Usern über die Interaktion follow folgen. Dadurch sehen die User dann den Content der gefolgten User auf einer speziellen Seite. Es handelt sich also um einen Inhaltsfilterungs bzw. Aggregationsprozess. Die User können hierdurch Einfluss auf die Inhalte, die sie präferiert sehen möchten, nehmen.

2.2 Social Network Analyse

Bei der Analyse sozialer Netzwerke geht es primär um die Interaktionsmuster zwischen den verschiedenen Akteuren in einem sozialen Netzwerk. Als soziales Netzwerk wird in diesem Kontext ein Zusammenschluss von Akteuren bezeichnet, die in einer oder mehreren Beziehungen zueinander stehen. Das soziale Netzwerk lässt sich aufgrund der Vernetzungen ideal als Graph darstellen. Hierbei werden die Akteure zu Knotenpunkten und die Beziehungen zwischen den Akteuren zu den Kanten des Graphen [MW11].
Ein Problem, dass bei der Social Network Analyse häufig auftritt, ist das so genannte Grenz-Spefizikations-Problem. Hierbei steht die Fragestellung, welche Akteure des Gesamtnetzwerks in die Analyse mit einbezogen werden sollen, im Vordergrund. Um dieses Problem zu lösen muss zunächst eine methodische Perspektive gewählt werden. Hierbei wird zwischen Realist und Nominalist unterschieden. Anschließend muss der Fokus der Abgrenzung gewählt werden. Hier gibt es 4 Möglichkeiten: Attributbasiert, beziehungsbasiert, eventbasiert sowie Multifokus. Anhand dieser Definition ergeben sich dann die Grenzen innerhalb des zu untersuchenden Teils des Netzwerks [LMP83].
Bei der Analyse sozialer Netzwerke werden eine Reihe von Kennzahlen ermittelt, die Aufschluss über das Netzwerk geben. Eine dieser Kennzahlen ist die Gradzentralität. Diese Kennzahl gibt an, wie viele Beziehungen ein Akteur besitzt. Somit werden die Akteure hervorgehoben, die für besonders viele andere Akteure von Bedeutung sind [KST09].
Eine weitere Kennzahl ist die Zwischenzentralität. Sie ist eine Weiterführung der Gradzentralität und gibt an, welcher Akteur für den Zusammenhalt des Netzwerks am wichtigsten ist. Dabei wird ausgewertet, wie viele direkte und indirekte Beziehungen ein Akteur ermöglicht. Dies wird erreicht, in dem betrachtet wird, wie viele Beziehungen insgesamt wegfallen, wenn der jeweilige Akteur wegfallen würde. Würde ein Gesamtnetz beispielsweise in viele kleine Netze zerfallen, hatte dieser Akteur eine hohe Zwischenzentralität [KST09].
Vo Relevanz ist weiterhin die Nähezentralität. Über diese Kennzahl wird angegeben, wie nahe ein Akteur durchschnittlich allen anderen Akteuren ist. Es wird also gemessen, über wie viele Knoten man mindestens von dem jeweiligen Akteur aus traversieren muss, um alle anderen Knotenpunkte erreicht zu haben [KST09].
Die KPI Dichte gibt Auskunft über den Vernetzungsgrad des Netzwerks. Hierbei wird die Anzahl an Beziehungen ins Verhältnis zur Anzahl der theoretisch möglichen Be-

Typ	**Anzahl**
Status	1099803
User	914257
Retweet	1415609
Reply	876276
Quote	345123

Tabelle 1: Anzahl der verschiedenen Rohdatensätze gruppiert nach Typ

ziehungen im Netzwerk gesetzt. Dieses Verhältnis gibt dann an, wie dicht, sprich wie vernetzt, das Netzwerk ist [KST09].
Mithilfe der Social Network Analyse können viele verschiedene Fragestellungen beantwortet werden. Abhängig von den vorliegenden Daten und mithilfe der ermittelten Kennzahlen können beispielsweise Rückschlüsse auf Befehlsstrukturen, Freundschaften oder Interessensgebiete gezogen werden. Weiterhin lassen sich Muster einfach erkennen und können genutzt werden, um beispielsweise Empfehlungsalgorithmen zu verbessern, indem die thematische Entfernung eines Users zu einem Thema innerhalb des Netzwerks berechnet wird.

2.3 Beschreibung des vorliegenden Datensatzes

Der vorliegende Datensatz liegt als ZIP-Archiv vor. Er ist also komprimiert. Innerhalb des Archivs befinden sich viele kleine Archive, die im GZIP-Format komprimiert sind. Diese beinhalten jeweils eine JSON-Datei, die dann eine Reihe von Rohdatensätzen enthält. Die inneren Archive sind nach Erstellungszeit sortiert. In Tabelle 1 ist zu sehen, wie viele Rohdatensätze jeweils pro Typ vorliegen.

3 Darstellung von Massendaten

Sollen Millionen von Datensätzen dargestellt werden, treten eine Reihe von Herausforderungen auf. Beispielsweise wäre es zu unübersichtlich und auch nicht performant genug, alle Datensätze gleichzeitig anzuzeigen. Wenn es nicht möglich ist, alle Datensätze gleichzeitig anzuzeigen, ist eine andere Möglichkeit, geeignete Aggregations-Algorithmen zu finden, die die bestehenden Daten beispielsweise gruppieren. Nach dieser Aggregation können die Daten dann aggregiert angezeigt werden, was die Anzahl der anzuzeigenden Daten verringert und somit die Übersichtlichkeit erhöhen kann [Elm+08].
Ein Problem bei diesem Ansatz ist, dass die Aggregations-Algorithmen abhängig vom Datensatz gewählt werden müssen, da diese eine semantische Bedeutung haben und somit nach fachlich sinnvollen Eigenschaften aggregiert werden muss. Geschieht das nicht, führt dies dazu, dass der Datensatz zwar überschaulicher aber auch weniger verständlich wird [Elm+08].
Eine andere Option ist, auf eine Aggregation zu verzichten, und nur einen Ausschnitt des Gesamtgraphen zu zeigen. Hierbei wird das Fenster, dass angezeigt wird, auf eine überschaubare Menge reduziert. Ein Nachteil dieser Methodik ist, dass es schwierig wird, einen Überblick über den gesamten Datensatz zu erhalten. Hierbei ist es dann oft hilfreich, relevante Kennzahlen, die den gesamten Datensatz beschreiben, zu ermitteln. Anhand dieser ist es ebenfalls möglich, einen übergreifenden Eindruck zu gewinnen.
Idealerweise kommt eine Kombination aus beiden Prinzipien zusammen. Diese Kombination wird auch als Fischaugen-Prinzip beschrieben. Es wird ein Einstiegspunkt in den Datensatz gewählt, der in höchster Auflösung, also nicht aggregiert, angezeigt wird. Je weiter ein Datensatz von diesem Einstiegspunkt entfernt ist, desto höher wird der Grad der Aggregierung. Somit ist es auch möglich, mehrere Aggregationen zeitgleich auf den Datensatz anzuwenden. Welche Aggregation gewählt wird hängt dann von der Entfernung des entsprechenden Punktes zum Einstiegspunkt des Graphen ab [Elm+08].
Wird nur ein Fenster angezeigt, ist es aber umso wichtiger, dass es eine Navigationsmöglichkeit durch den Datensatz gibt. Das Fenster muss also verschoben werden können. Um dies zu erreichen, muss der Einstiegspunkt in den Gesamtdatensatz neu gewählt werden. Anhand des neu gewählten Einstiegspunktes ergeben sich dann die neuen Aggregationszonen [PFW00].

3.1 Darstellung als Netzwerk

Stark vernetzte Massendaten lassen sich als Netzwerk darstellen. Dabei besteht so ein Netzwerk aus Knotenpunkten und Kanten. In der Regel repräsentiert jeder Knotenpunkt einen Datensatz innerhalb der Massendaten und jede Kante stellt eine Beziehung zwischen zwei Datensätzen her. Dabei kann das Netzwerk entweder in 2D oder in 3D

dargestellt werden. Jedem Knotenpunkt wird dann eine Position in dem entsprechenden Raum zugeordnet. Anschließend können die Kanten durch Vektoren zwischen den jeweiligen Punkten realisiert werden.
Will man nun ein Netzwerk von Millionen von Knotenpunkten und Kanten visualisieren und intuitiv verständlich machen treten die bereits beschriebenen Probleme auf. Daher muss entweder aggregiert oder mithilfe eines Fensters gearbeitet werden, da es sonst nicht möglich ist, mit dem Datensatz zu arbeiten.
Eine Möglichkeit, Übersichtlichkeit im Netzwerk zu erzeugen, ist, die verschiedenen Knotenpunkte und Kanten sich in einer oder mehrerer Dimensionen unterscheiden zu lassen. Beispielsweise könnte die Anzahl an Kanten, die ein Knotenpunkt besistzt, durch seine Größe repräsentiert werden. Dadurch kann ein User deutlich leichter bewerten, welche Knoten in der entsprechenden Dimension welche grobe Ausprägung haben und kann die verschiedenen Knotenpunkte leicht in ein Verhältnis zueinander setzen.
Dieses Prinzip kann auch genutzt werden, um textuelle Label im Graphen zu verringern. So kann der Typ einer Relation beispielsweise durch die Farbe anstelle eines Labels an der Kante repräsentiert werden. Dies führt gerade bei der Darstellung von Massendaten, deren Elemente der Darstellung durch die schiere Menge an Daten häufig so klein sind, dass Text nicht mehr gelesen werden kann, dazu, dass über die Farbe nach wie vor ein Überblick über die Verhältnismäßigkeiten im Netzwerk möglich ist.
Ein zentrales Problem bei der Darstellung von Netzwerken ist das Layout. In welcher Form sollen die Knotenpunkte angeordnet werden? Da die Netzwerke häufig viel zu groß sind, um im Vorfeld ein optimales Layout in akzeptabler Zeit zu berechnen, werden vielfach iterative Algorithmen genutzt, die stückweise eine Annährerung an ein optimales Layout schaffen.
Viele dieser Algorithmen basieren dabei auf einer Simulation physischer Kräfte wie beispielsweise Magnetkraft, sowohl Anziehung als auch Abstoßung, Spannung oder Reibung. Dabei basieren diese Kräfte häufig auf den Attributen einzelner Knotenpunkte, beispielsweise kann die Größe eines Knotenpunktes Einfluss auf seine Anziehungskraft haben oder die Länge einer Kante auf die Spannung zwischen den Knotenpunkten, die über die Kante verbunden sind.
Ein Nachteil von Layouts, die auf der Simulation von Kräften basieren, ist die in der Regel hohe Laufzeit, gerade bei Massendaten. Wenn nicht im Vorfeld aggregiert werden kann, müssen andere Performance-Optimierungen durchgeführt werden, um auch bei mehr als 100.000 Knotenpunkten noch eine akzeptable Performance zu erhalten. Eine dieser Optimierungen ist die Barnes-Hut-Simulationstechnik, bei der die Knotenpunkte anhand ihrer Position im Graphen gruppiert werden. Anschließend können die Kräfte innerhalb der Gruppe anhand der Knotenpunkte und außerhalb der Gruppe zwischen den Gruppen berechnet werden, was eine deutliche Verringerung der Laufzeit bedeuten kann [BH86].

3.2 Darstellung als Matrix

Anstelle einer Dartstellung als Netzwerk lassen sich stark verknüpfte Massendaten auch als Matrix darstellen. Hierbei werden die verschiedenen Datensätze jeweils einmal in die x- und einmal in die y-Achse einer Matrix eingetragen. Gibt es nun eine Verbindung zwischen 2 Datensätzen wird diese Zelle, also der Schnittpunkt beider Datensätze, innerhalb der Matrix entsprechend gekennzeichnet.[Elm+08]
Ein großer Nachteil an dieser Layout-Methode ist, dass es für jede theoretische Verbindung innerhalb des Datensatzes eine Zelle gibt. Diese ist entweder gefüllt, wenn die Verbindung auch tatsächlich besteht, oder, falls nicht, entsprechend anders gekennzeichnet. Es muss jedoch eine Kennzeichnung erfolgen, da die Daten ja in eine Matrix eingetragen werden. Somit muss ein User nicht nur die Stellen sehen, an denen es eine tatsächliche Verbindung von Daten im Datensatz gibt, sondern auch alle, an denen es keine gibt. Dies ist mit unter eine enorme Verschwendung an Platz und führt dazu, dass der User von den Inhalten, die ihn eigentlich interessieren, abgelenkt wird.
Ein weiterer Nachteil ist, dass es ohne Aggregation der Daten unmöglich ist, einen Überblick über die Daten zu erhalten, wenn der Datensatz mehr als einige hundert Daten enthält, während das bei der Netzwerkdarstellung noch kein Problem ist. Insofern passt das Konzept der Matrixdarstellung nicht zu dem vorliegenden Datensatz, der Millionen von Daten enthält.

3.3 Interaktivität der Darstellung

Die Datengrundlage für eine Netzwerkdarstellung unterliegt einer extremen Varianz. Die anzuzeigenden Daten können sich beispielsweise in Anzahl, Vernetzungsgrad sowie der Varianz des Vernetzungsgrades, also wie groß der Unterschied zwischen des am vernetztesten und am wenigsten vernetztesten Knotenpunkt ist, massiv unterscheiden. Will man diese Daten nun als Netzwerk anzeigen, macht dies die Auswahl des korrekten Layouts sehr schwierig. Selbst mit sehr generischen Layouting-Algorithmen kann es nach wie vor passieren, dass bestimmte Stellen im Graphen suboptimal gelayouted wurden.
Hier bietet Interaktivität mit dem Graphen einen großen Mehrwert. Anstatt im Vorfeld das Ziel zu haben, den Graphen in jedem Fall perfekt zu layouten, was wie bereits aufgezeigt neben den diversen Herausforderungen auch zu erheblichen Performance-Problemen führen kann, kann der User durch Interaktivität nachträglich Detailveränderungen an dem Graphen vornehmen, um seinem aktuellen Anspruch gerecht zu werden. Hierbei gibt es eine Reihe von Parametern, mit denen ein User arbeiten kann.

3.3.1 Kräfte

Wenn das Netzwerk-Layout einer Kräfte-Simulation entspricht, ist es möglich, kann dem User die Möglichkeit gegeben werden, Einfluss auf die wirkenden Kräfte, ihre Ausprägung und Berechnung zu nehmen. Weiterhin ist es möglich, bestimmte Presets an Kräftekombinationen zu definieren, zwischen denen ein User dann leicht wählen kann. Somit kann ein User dann den vorliegenden Datensatz leicht in verschiedenen Layouts wie Fruchterman-Reingold [FR91] oder ForceAtlas2 [Jac+14] zu betrachten. Somit kann die Entscheidung, welcher Layouting-Algorithmus verwendet werden soll, dem User überlassen werden. Da neben objektiven Kriterien auch subjektive Wahrnehmung und Ästhetik eine Rolle bei der Layout-Auswahl eine Rolle spielt, ist dies auch sehr wichtig.
Eine Herausforderung ist hierbei, dass das Layout nicht im Vorfeld berechnet werden kann, sondern auf Userwunsch mit den gewählten Parametern in Echtzeit brechnet werden muss. Da das Layouting, abhängig von der konkreten Datengrundlage, sehr viel Zeit in Anspruch nehmen kann, muss der User entweder gewillt sein, eine gewisse Zeit auf das berechnete Layout zu warten, oder ein suboptimales Layout durch vorzeitigen Abbruch der Simulation in Kauf zu nehmen.

3.3.2 Feinjustierung des Layouts

Ist das generelle Layout erstellt und wird dem User angezeigt, kann es sein, dass es bestimmte Stellen im Graphen gibt, die der User ein wenig anders angezeigt haben möchte. Handelt es sich dabei nur um die Position einzelner Knotenpunkte, ist es beispielsweise möglich, dass der User diese Position selbstständig via Drag&Drop anpasst. Wurde ein Kräfte-basiertes Layout genutzt sollte die zu Grunde liegende Simulation während des Drag&Drop-Vorganges kontinuierlich weiterlaufen, wenn dies aus Performance-Sicht noch vertretbar ist, um weiterhin eine homogene Darstellung zu erzielen. Wird die Simulation nicht weitergeführt, stechen die Modifikationen durch den User in der Regel deutlich aus dem Layout hervor, da sie nicht integriert wirken.

3.3.3 Verschieben des Sichtfensters

Wie bereits beschrieben werden Netzwerke in der Regel in einen zweidimensionalen Raum projeziert. Wird nun ein Teilausschnitt des Gesamtnetzwerks angezeigt, ist dieser Anzeige eine bestimmte Position in der Gesamtebene, die durch den 2D-Raum definiert ist, zugeordnet. Ermöglicht man es dem User jetzt, diese Position zu verschieben, kann der User selbstständig den aktuellen Ausschnitt des Netzwerks, den er sieht, wählen. Somit kann er sich interaktiv durch das Netzwerk bewegen.

3.3.4 Zoomen

Wird ein komplexes Netzwerk komplett angezeigt, dem ein großer Datensatz zu Grunde liegt, sind die jeweiligen Knotenpunkte und Kanten mitunter sehr klein. Will ein User jetzt einen bestimmten Teil des Netzwerks genauer untersuchen, bietet es sich an, diesen Ausschnitt zu vergrößern. Ermöglicht man es dem User jetzt, sich durch ein Vergrößern und Verkleinern des Bildausschnittes innerhalb des Netzwerks zu bewegen, führt dies gerade in Kombination mit einer möglichen Verschiebung der Position innerhalb des Netzwerks dazu, dass ein User jederzeit genau den gewünschten Ausschnitt des Netzwerks in der gewünschten Auflösung betrachten kann.

3.3.5 Kontextsensitivität

Interaktivität bietet einen weiteren Vorteil, um den Graphen deutlich übersichtlicher zu machen. Häufig enthalten die Knotenpunkte im Graphen eine ganze Reihe von Attributen, die von potentiellem Interesse für den User sind. Werden diese jedoch fest in die Darstellung gerendered, führt dies neben dem deutlich größerem Platzbedarf auch zu deutlich mehr Informationen, die ein User zeitgleich wahrnimmt. Insofern leidet die Übersichtlichkeit unter diesen Informationen, gerade weil häufig nur einen Bruchteil aktuell für den User von Relevanz ist.
Um dieses Problem zu lösen, können diese Informationen beispielsweise nur bei Bedarf angezeigt werden, beispielsweise wenn ein User mit der Maus über einen Knotenpunkt fährt oder diesen anklickt.
Es gibt einen weiteren Interaktionsmechanismus, der die Übersichtlichkeit verbessert. So ist es beispielsweise möglich, wenn ein User mit einem Knotenpunkt interagiert, alle Knotenpunkte und Kanten auszugrauen, die nicht in direkter Relation zu ebenjenem Knotenpunkt stehen. Dadurch kann ein User auch in komplexen Layouts einfach identifizieren, welche Knotenpunkte in welcher Relation zu einem gewählten Knotenpunkt stehen.

4 Anforderungsanalyse

Dieses Kapitel beschreibt die vorgenommene Anforderungsanalyse. Um die zentrale Frage zu beantworten, wie der vorliegende Datensatz durchsuchbar gemacht und visualisiert werden kann, wurden im Vorfeld die Anforderungen, die dafür erfüllt sein müssen, ermittelt.

4.1 Funktionale Anforderungen

Die nachfolgenden funktionalen Anforderungen sind aus der Forschungsfrage dieser Arbeit sowie der vorhergegangenen Literaturrecherche abgeleitet.

4.1.1 Daten-Upload (F1)

Da die zu konzipierende Softwarelösung mit dem bestehenden Datensatz arbeiten und nicht direkt die Twitter-API konsumieren soll, sollte es eine einfache Möglichkeit für die User der Software geben, einen Datensatz, der untersucht werden soll, hochzuladen. Hierfür wird eine Upload-Funktion benötigt, über die die User über das Frontend ihre Datei auswählen und diese zum Server übertragen können.

4.1.2 Quantitative Aspekte des Datensatzes (F2)

Damit die User einen Überblick über die bereits importieren Daten haben, sollen verschiedene KPIs über das Frontend zugänglich gemacht werden:

- Die Anzahl an importierten Rohdaten je Typ (F2.1)
- Die Anzahl an Knotenpunkten im Netzwerk je Typ (F2.2)
- Die Anzahl an Kanten im Netzwerk pro Kantentyp (F2.3)
- Die Anzahl an Interaktionen eines Users pro Interaktion (F2.4)
- Die Varianz der Interaktionshäufigkeit der User (F2.5)

4.1.3 Zurücksetzen der Applikation (F3)

Damit ein User mit einem neuen Datensatz arbeiten kann muss er die Applikation zurücksetzen können. Dafür sollten sämtliche Daten aus den Datenbanken gelöscht werden. Dies soll ihm einfach über das Frontend möglich sein.

4.1.4 Steuerung/ Administration der Plattform (F4)

Die Software soll sich durch den User steuern lassen. Dafür ist es unablässig, dass eine Funktionalität entwickelt wird, über die der User bestimmen kann, wann welche

Aspekte der Software ausgeführt werden. Dieser Bedarf wird dadurch verstärkt, dass einige der Datenimportprozesse lange laufen können und das während dieser Zeit eine erhöhte Last auf dem System vorliegt. Damit diese Last den User nicht bei seiner Arbeit behindert, soll es möglich sein, dass der User diese Prozesse dann ausführen lässt, wenn er die Software gerade nicht bedient.

4.1.5 Information über Fortschritt verschiedener Datenimport-Prozesse (F5)

Da die Datenimportprozesse teilweise sehr lange dauern können ist es für den User wichtig, über den Status dieser Prozesse informiert zu sein. Daher soll die Möglichkeit geschaffen werden, dass sich ein User selbstständig über Zustand sowie Fortschritt der diversen nebenläufigen Prozesse der Applikation informieren kann.

4.1.6 Suchmaschine (F6)

Damit es einem User möglich ist, in dem riesigen Netzwerk einen Einstiegspunkt zu finden, soll es die Möglichkeit geben, mithilfe mehrerer Faktoren zu suchen. Beispielsweise soll es möglich sein, dass über das Twitter-Handle eines Twitter-Users gesucht werden kann. Weiterhin soll es möglich sein, via Hashtags zu filtern.

4.1.7 Darstellung des Datensatzes als interaktives Netzwerk (F7)

Das Kernstück der Software soll die Darstellung des vorliegenden Datensatzes als interaktives Netzwerk sein. Da in dem vorliegenden Datensatz potentiell zu viele Knotenpunkte und Kanten vorliegen, als das diese noch anschaulich und verständlich visualisiert werden können, soll es möglich sein, die Anzahl an geladenen Knotenpunkten zu begrenzen. (F7.1)
Weiterhin ist aufgrund der potentiell großen Varianz der Kanten, die von einem Knoten des Netzwerks ausgesehen, wichtig, dass der User auf die Darstellung des Netzwerkes Einfluss nehmen kann, um so potentiell unanschauliche Darstellungen des gewählten Ausschnitts des Netzwerkes trotzdem verstehen zu können. Hierfür soll es eine Möglichkeit geben, dass der User die Parameter, die der Darstellung zu Grunde liegen, verändern kann. Weiterhin soll es möglich sein, via Drag-and-Drop Feinjustierungen der Positionierung einzelner Knotenpunkte vorzunehmen. (F7.2)
Damit ein User sich über die verschiedenen Daten eines Knotenpunktes bei Bedarf informieren kann, soll es möglich sein, diese Informationen dynamisch ein- und auszublenden. Hierfür soll die Hover-Interaktion genutzt werden, also wenn ein User mit der Maus über einen Knotenpunkt fährt, sollen die relevanten Informationen zu diesem Knotenpunkt eingeblendet werden. (F7.3)
Weiterhin soll es möglich sein, dass ein User mit einem Blick auf das Netzwerk potentiell relevante Knotenpunkte identifizieren kann. Um dies zu realisieren sollen sich die Knotenpunkte in 2 Dimensionen unterscheiden können: Größe und Farbe. Somit kann

ein User über die Ausprägung dieser beiden Dimensionen in einem Knotenpunkt direkt entscheiden, ob dieser relevante Informationen beinhalten könnte. (F7.4)
Um sich schnell und effektiv durch das Netzwerk bewegen zu können soll es möglich sein, dass ein User durch einen simplen Mechanismus einen gewählten Knotenpunkt als neues Zentrum des dargestellten Graph-Ausschnittes benennen kann. Dies ermöglicht eine schnelle und intuitive Navigation durch den Graphen. Weiterhin soll es möglich sein, dass ein User durch eine Zoomfunktion das sichtbare Fenster des Graphen verändern kann. In Kombination dazu soll er auch die Möglichkeit haben, sich innerhalb des angezeigten Graphen zu bewegen, indem er das angezeigte Fenster verschiebt. (F7.5)

4.2 Technische Anforderungen

Aus den genannten funktionalen Anforderungen ergeben sich einige technische Anforderungen, die Vorraussetzung für die Erfüllung der funktionalen Anforderungen sind.

4.2.1 Skalierbarer Datenimport (T1)

Um es zu ermöglichen, auch eine große Anzahl an Datensätzen schnell einlesen zu können, soll es möglich sein, den Datenimportprozess horizontal skalieren zu können. Dies bedeutet, dass es möglich sein soll, mehrere Instanzen der Anwendung auf unterschiedlichen Hosts gleichzeitig zu betreiben und jeder Anwendung eine Teilmenge der zu importierenden Daten zu übergeben. Diese Skalierung betrifft nur die Applikation selbst, nicht die unterliegende Datenbank. Die Skalierung ist somit durch die Datenbank limitiert. Diese Limitierung ist aber als in Ordnung bewertet worden, da Datenbanken in der Regel deutlich performanter als die eigentliche Applikation sind. Weiterhin ist es auch möglich, die Datenbanken bis zu einem gewissen Level durch das Bilden eines Datenbank-Clusters inkl. Shardings zu skalieren.

4.2.2 Parallelbetrieb Datenimport und Datenanalyse (T2)

Um es einem User zu ermöglichen, das Frontend der Applikation unabhängig von der erzeugten Last aus einem Datenimportprozess zu nutzen, soll eine klare Trennung zwischen dem Frontend-Teil der Applikation und dem Datenimport-Teil vorliegen. Diese Trennung stellt sicher, dass die Komponenten unabhängig voneinander skaliert werden können. Weiterhin ermöglicht diese Microservice-Architektur schnellere Deployments und ein ausfallsichereres System, denn ein Problem in einem der beiden Komponenten führt nicht zu einem Ausfall der jeweils anderen Komponente.

4.2.3 Betrieb als Docker-Container (T3)

Um einen schnellen und unkomplizierten Betrieb der Anwendung inklusive ihrer Abhängigkeiten zu ermöglichen soll ein Dockerfile erstellt werden, dass einen Container beschreibt, in dem die Anwendung mit allen Features und Abhängigkeiten läuft.

5 Architektur

Aufgrund der Erkenntnisse, die in Unterabschnitt 5.1 gewonnen wurden, wurde sich für eine Client-Server-basierte Architektur entschieden. Hierbei wurde der Browser als Client und somit eine klassische Architektur einer Webapp gewählt. Die eigentliche Anwendung sollte in 3 Microservices aufgeteilt werden. Diese werden im Folgenden beschrieben.

5.1 Thin Clients vs Fat Clients

Software wird häufig in 2 Teile aufgeteilt: Backend und Frontend. Es gibt also eine klare Trennung zwischen dem Teil einer Applikation, die primär für Datenimport- und Verarbeitungsprozesse zuständig ist und dem Teil, der für die Anzeige bestimmter Daten zuständig ist. Diese Trennung kann sich von nur im Code bis hin zur kompletten Systemlandschaft ziehen.
Häufig benötigt der Backend-Teil der Software eine ganze Reihe von weiteren Komponenten, um korrekt zu funktionieren. Das kann eine Laufzeitumgebung wie Java sein oder weitere Programme, mit denen über ein Protokoll wie HTTP kommuniziert wird. Außerdem beinhaltet der Backend-Teil häufig Algorithmen oder andere spezifische Implementierungsdetails, die der Produzent des Programms geheim halten möchte.
Daher bietet es sich an, dass zumindest der Backend-Teil direkt bei dem Produzenten der Software ausgeführt wird und nur der Frontend-Teil an einen User ausgeliefert wird. Aber häufig benötigt auch der Frontend-Teil eine ganze Reihe von Komponenten, um zu funktionieren. Da das Frontend aber zum User ausgeliefert wird, hat der Produzent nur einen gewissen Einfluss auf das Ökosystem, in dem sein Frontend betrieben wird. Daher ist es von Vorteil, die benötigten Abhängigkeiten zu minimieren und den Teil, der benötigt wird, mit auszuliefern.
Die momentan stärkste Form dieser Minimierung ist ein Web-Browser. Dieser ist auf einem Großteil aller Computer, die momentan in Benutzung sind, vorinstalliert bzw. werden vom User installiert, da er sowieso einen Bedarf für eine Web-Browser hat.
Ein Nachteil daran, den Browser als Frontend zu nutzen, ist die eingeschränkte Leistung. Eine Desktop-Applikation kann durch multithreading und eine bessere Nutzung der Grafikkarte deutliche Performancevorteile im Vergleich zu Web-Applikationen im Browser erzielen. Jedoch ist die Performance von Javascript im Browser für die meisten Anwendungsfälle mehr als ausreichend und durch bestimmte Optimierungen können auch komplexe Anwendungen wie beispielsweise Bildbearbeitungssoftware oder Office-Programme im Browser flüssig nutzbar gemacht werden.

5.2 Technologieauswahl

Da es sich bei der zu konzipierenden Software um eine komplexe Anwendung mit einer Vielzahl verschiedener Features handelt, sollen Teile der Anwendung durch bereits bestehende Technologie abgedeckt werden. im Folgenden wird aufgezeigt, welche Technologien eingesetzt werden und was die Gründe dafür sind.

5.2.1 Datenbanken

Das Kernstück der Anwendung ist die Visualisierung der Twitter-Daten. Diese Daten haben eine hochgradige Vernetzung untereinander. Daher eignet sich der Einsatz einer Graphdatenbank, um die Daten durchsuchbar und traversierbar zu machen. Weiterhin ist durch eine Graphdatenbank ein simples Datenmodell nutzbar.
Weiterhin müssen die Rohdaten gespeichert werden. Diese liegen erstmal als Zip-Archiv vor, dass mehrfach dekomprimiert werden muss, um an die eigentlichen Daten zu kommen. Weiterhin lässt sich das Archiv auf keinster Weise durchsuchen. Die Daten innerhalb der verschiedenen JSON-Dateien liegen weiterhin unstrukturiert vor.
Die programmatische Arbeit mit dem Zip-Archiv ist also mühsam, komplex und die Datenstruktur ermöglicht keinen effizienten Zugriff auf einzelne Datensätze. Aus den Anforderungen an den Datenimportprozess, auf die genauer in Unterabschnitt 6.7 eingegangen wird, ergibt sich die klare Anforderung, eine Datenbank für die Persistierung der Rohdaten einzusetzen. Da die Rohdaten in Dokumentenstruktur vorliegen bietet sich der Einsatz einer dokumentenbasierenden Datenbank an. Aufgrund der hohen Performance, Flexibilität und Vielseitigkeit wurde die Mongo-DB ausgewählt. Sie bietet unter Anderem mit der Aggregation-Pipeline einen hochgradig effizienten und machtvollen Weg, Insights aus den vorhandenen Daten zu gewinnen. Dieser Vorteil kommt besonders zu tragen, wenn quantitative Fragestellungen über den Rohdatensatz beantwortet werden sollen, was ja ein Teil der Anforderungen ist. Weiterhin gibt es bei dem Import der Rohdaten keinen Bedarf für Transaktionen, was eines der größten Vorteile einer SQL-basierten Datenbank wäre [Gyö+15].
Da die Daten in einer stark vernetzten Struktur vorlieren, bietet sich die Nutzung einer Graphdatenbank an. Bei der Wahl der Graphdatenbank fiel die Wahl auf neo4j. Diese entscheidung wurde vor allem aufgrund der Popularität von neo4j getroffen. Die Anforderungen, die diese Applikation an eine Graphdatenbank stellt, werden durch neo4j erfüllt. Auf dem Popularitätsindex für Graphdatenbanken liegt neo4j abgeschlagen auf Platz 1 [DB-20]. Da Platz 2 an die Cloud-only Datenbank Microsoft Azure Cosmos DB geht, die im Rahmen dieser Arbeit aufgrund der erhöhten Komplexität durch eine Cloud-native-Lösung ohnehin nicht hätte untersucht werden können, ist die nächste Datenbank, der Platz 3, ArangoDB, mit nur noch einem Zehntel der Beliebtheit von neo4j deutlich abgeschlagen. Die nächste reine Graphdatenbank, JanusGraph, hat nur

noch 3,4% der Popularität von neo4j. Insofern wird daraus abgeleitet, dass es sich bei neo4j um den deutlichen Industrie-Anführer handelt.

5.2.2 Spring

Für die Webapp sollte Spring genutzt werden. Spring bietet ein komplexes Ökosystem an Technologien, die rund um eine Webapp benötigt werden. Einige Beispielse hierfür sind Dependency Injection, Data-Binding, Integration mit einem Application Server mit spring-webmvc sowie Zugriff auf Datenbanken via spring-data. Die benötigten Treiber, um auf die Mongo-DB sowie die neo4j-Graphdatenbank zuzugreifen, sind über spring-data nutzbar.

5.2.3 Frontend

Aufgrund der hohen Komplexität des Frontends soll die GUI als Single-Page-Application (SPA) erstellt werden. Bekannte SPA-Frameworks wie vue.js, Angular oder React bieten dann Unterstützung bei den Themen Routing, State Management sowie Reactivity. Weiterhin führt der komponentenbasierte Ansatz dieser Frameworks zu einer saubereren Frontend-Architektur.
Für das Rendern des Graphen sollte ebenfalls ein Framework genutzt werden. Von den populärsten Lösungen decken nur d3.js und sigma.js sämtliche Anforderungen komplett ab. So bietet beispielsweise chart.js gar keine Möglichkeit, ein Netzwerk zu rendern. Vergleicht man die Popularität von d3.js [npm20a] mit der von sigma.js [npm20b] wird ersichtlich, dass sigma.js nur einen Bruchteil an Popularität, gemessen an den wöchentlichen Downloads, von d3.js besitzt. Weiterhin ist das letzte Release von sigma.js vom 13.10.2017, das letzte Release von d3.js vom 29.12.2019. Insofern scheint d3.js die bessere Wahl, da die Community aktiver ist und die Technologie aktiv weiterentwickelt wird.

5.3 Datenmodell

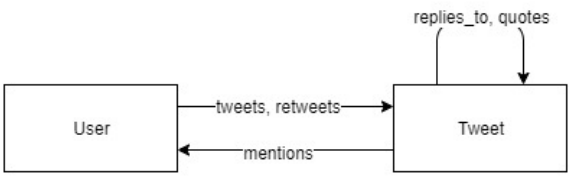

Abbildung 1: Visualisierung des Datenmodells des Graphen

In Abbildung 1 ist das Datenmodell dargestellt, dass dem Graphen in der Graphdatenbank zu Grunde liegt. Wie zu sehen ist, gibt es zwei Entitäten, User und Tweet. Diese Entitäten können in einer Reihe von Relationen zu einander stehen. So können

User Tweets anlegen (tweets) und teilen (retweets), Tweets können andere User erwähnen (mentions) sowie auf andere Tweets antworten (replies_to) sowie zitieren (quotes). Dieses Datenmodell kann somit alle in Unterabschnitt 2.1 aufgezeigten Interaktionen abbilden, die sich aus dem vorliegenden Datensatz ableiten lassen.

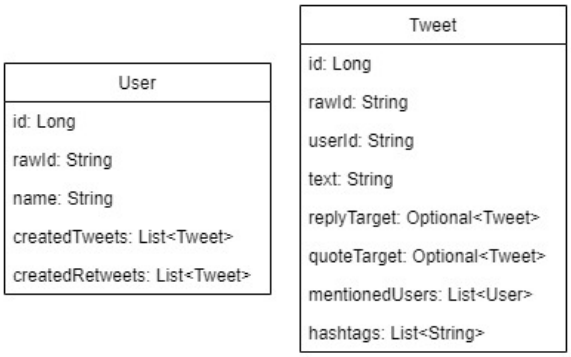

Abbildung 2: Klassendiagramm der Entitäten User und Tweet

Abbildung 2 zeigt die Felder der Entitäten. Es folgt eine kurze Erläuterung der Felder.

User

- id: Die Datenbank-ID in der Graphdatenbank. Wird von neo4j gewählt
- rawId: Verweis auf die ursprüngliche Id, die dieser User bei Twitter hatte
- name: Der Username des Users
- createdTweets: Eine Liste mit Verweisen auf die Tweets, die dieser User angelegt hat
- createdRetweets: Eine Liste mit Verweisen auf die Tweets, die dieser User geteilt hat

Tweet

- id: Die Datenbank-ID in der Graphdatenbank. Wird von neo4j gewählt
- rawId: Verweis auf die ursprüngliche ID, die dieser Tweet bei Twitter hatte
- userId: Die ursprüngliche Twitter-ID des Autors
- text: Der textuelle Inhalt des Tweets
- replyTarget: Falls gesetzt, ein Verweis auf den Tweet, auf den Dieser antwortet
- quoteTarget: Falls gesetzt, ein Verweis auf den Tweet, den Dieser zitiert

- mentionedUsers: Eine Liste mit Verweisen auf die User, die von diesem Tweet erwähnt werden
- hashtags: Eine Liste mit den in diesem Tweet verwendeten Hashtags

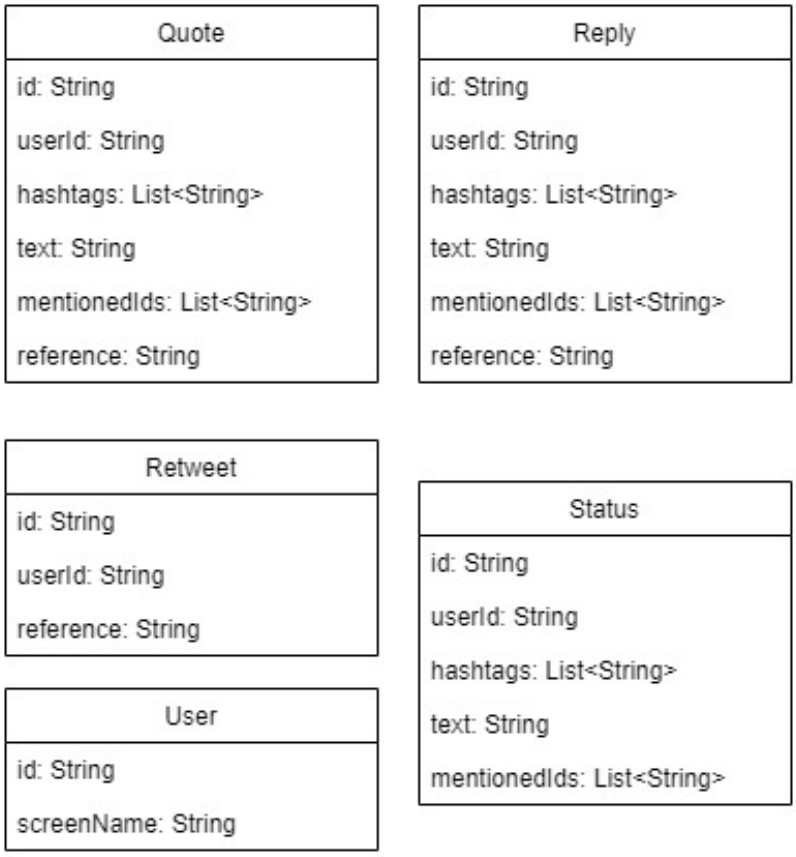

Abbildung 3: Klassendiagramm der verschiedenen Entitäten der Rohdaten

Für die Rohdaten ist ein Datenmodell in Abbildung 3 zu finden. Die Erklärung der Felder ist aus der Erklärung des Graph-Datenmodells ableitbar, daher wird an dieser Stelle auf eine erneute Erklärung verzichtet.

5.4 System-Übersicht

Ganz im Sinne des Single-Responsibility-Principles und der 12-Factor-App folgend sollte die Gesamtapplikation in mehrere Microservices aufgespalten werden, um unabhängige Skalierbarkeit und Deployments zu ermöglichen. Die grundlegende Aufteilung spaltet die Gesamtapplikation in drei Microservices auf. Dies ist in Abbildung 4 aufgezeigt.

5.5 Frontend-Architektur

Bestimmte Teile des Frontends nutzen, wie in der Übersicht ja ersichtlich, Schnittstellen des Tweetalyzers und andere Schnittstellen des Tweetimporters. Insofern ist das

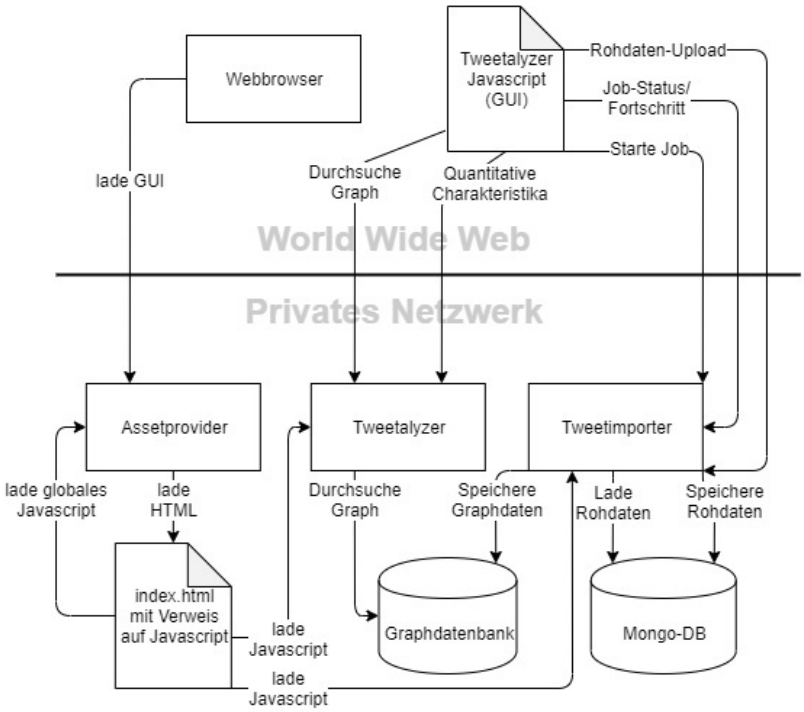

Abbildung 4: Übersicht über die grundlegende Systemarchitektur

Frontend lediglich eine Weiterführung der Verantwortung dieser Microservices. Daher müssen auch die Teile des Frontends, die zur jeweiligen Domäne gehören, auch zu dem jeweiligen Microservice zugeordnet werden [Jac19]. Dies führt zu der Herausforderung, die verschiedenen Frontend-Komponenten in korrekter Weise zusammenzuführen. Hierfür sollen npm-Pakete genutzt werden. Dabei handelt es sich um ein System zur Verwaltung von Javascript-Abhängigkeiten, äquivalent zu Maven oder Gradle bei Java. Die einzelnen Frontend-Komponenten sollen also mithilfe von npm zu so genannten Packages gebündelt werden, die dann wiederum von allen anderen Teilen des Frontends importierbar sind.

Dies führt dann zu der groben Komponenten-Aufteilung in Abbildung 5. Wie zu sehen ist, wird die GUI in zwei Seiten aufgeteilt: Die Graph-Seite und die Admin-Seite. Auf der Graph-Seite soll der gerenderte Graph einsehbar sein, inklusive der Suche und sämtlichen Steuerungsmöglichkeiten, um den Graph zu manipulieren. Auf der Admin-Seite findet sich die Job-Übersicht inklusive Steuerung und die quantitativen Charakteristika der Roh- und Graphdaten. Diese Seiten müssen jeweils einem Microservice zugeordnet werden. Aufgrund der Verantwortlichkeiten der jeweiligen Seiten wird die Graph-Seite dem Tweetalyzer und die Admin-Seite dem Tweetimporter zugeordnet. Hierbei muss beachtet werden, dass die Seitenaufteilung in der GUI nicht anhand der Systemarchitektur durchgeführt werden sollte. Ein nach UX-Aspekten designter User-Flow durch die Applikation orientiert sich an anderen Maßstäben als die Einteilung der Systemarchitektur in Domänen.

Wie zu sehen ist, gibt es auch globale Komponenten, die für alle Seiten gebraucht

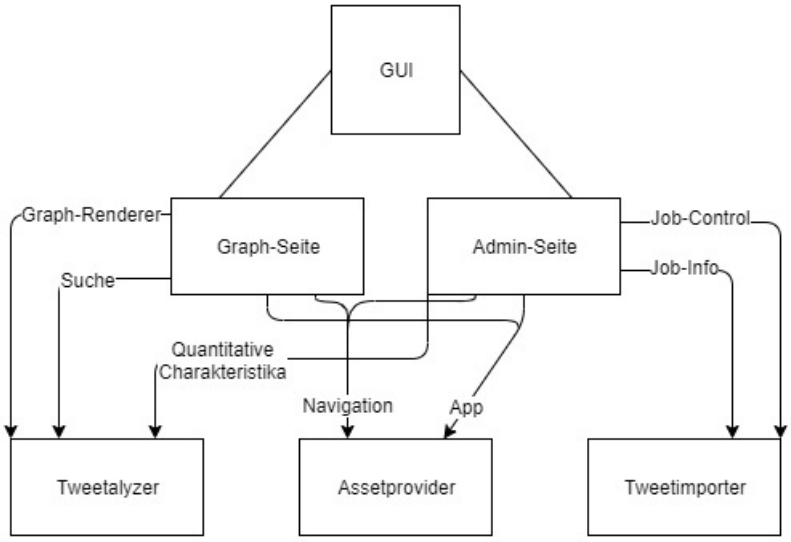

Abbildung 5: Übersicht über die Frontend-Architektur

werden, wie die Navigation oder die grundsätzliche App im SPA-Kontext. Diese Komponenten werden vom Assetprovider ausgeliefert, könnten aber auch in einen extra Microservice ausgelagert werden. Da dieser aber weder Backend-Logik noch eine Datenbank benötigen würde, wurde darauf verzichtet, um die Komplexität so gering wie möglich zu halten.

Um einen besseren Überblick über den gesamten Prozess zu gewinnen, wurde in Abbildung 6 einmal visualisiert, wie ein möglicher Userflow durch die Anwendung abläuft. Wie zu sehen ist, beginnt alles damit, dass der User eine der Seiten ansteuert, in diesem Fall die Graph-Seite (1). Der Asset-Provider liefert für alle Anfragen nach bestimmten Seiten die index.html-Datei aus, in der eine Referenz auf die App-Komponente vorhanden ist. Der Browser lädt also die index.html-Datei (2). Anschließend löst er die Referenz auf die App-Komponente auf und lädt diese (3). Daraufhin wird diese initialisiert (4). Da die App-Komponente die Navigation-Komponente beinhaltet, wird auch diese initialisiert (5). Das Beinhalten wird während des Bundlings realisiert. In diesem Prozessschritt des CI-Prozesses werden Abhängigkeiten unter den Javascript-Komponenten aufgelöst. Das Ergebnis ist eine Javascript-Datei, die alle Abhängigkeiten transpiliert beinhaltet.

Damit nicht die komplette Applikation inklusive aller Seiten beim initialen Seitenaufruf geladen werden muss, sollte immer nur die für die aktuell benötigten Komponenten geladen werden. Dies wird über dynamische Abhängigkeiten in der Navigation-Komponente realisiert, die erst zur Laufzeit aufgelöst werden. Da der User initial auf die Graph-Seite navigiert hat, löst die Navigation-Komponente jetzt diese Abhängigkeit (6) auf und somit lädt der Browser die GraphPage-Komponente vom Asset-Provider (7). Diese Komponente wird anschließend initialisiert (8). Sie enthält auch weitere Sub-Komponenten, die GraphRenderer-Komponente sowie die Search-Komponente. Auch

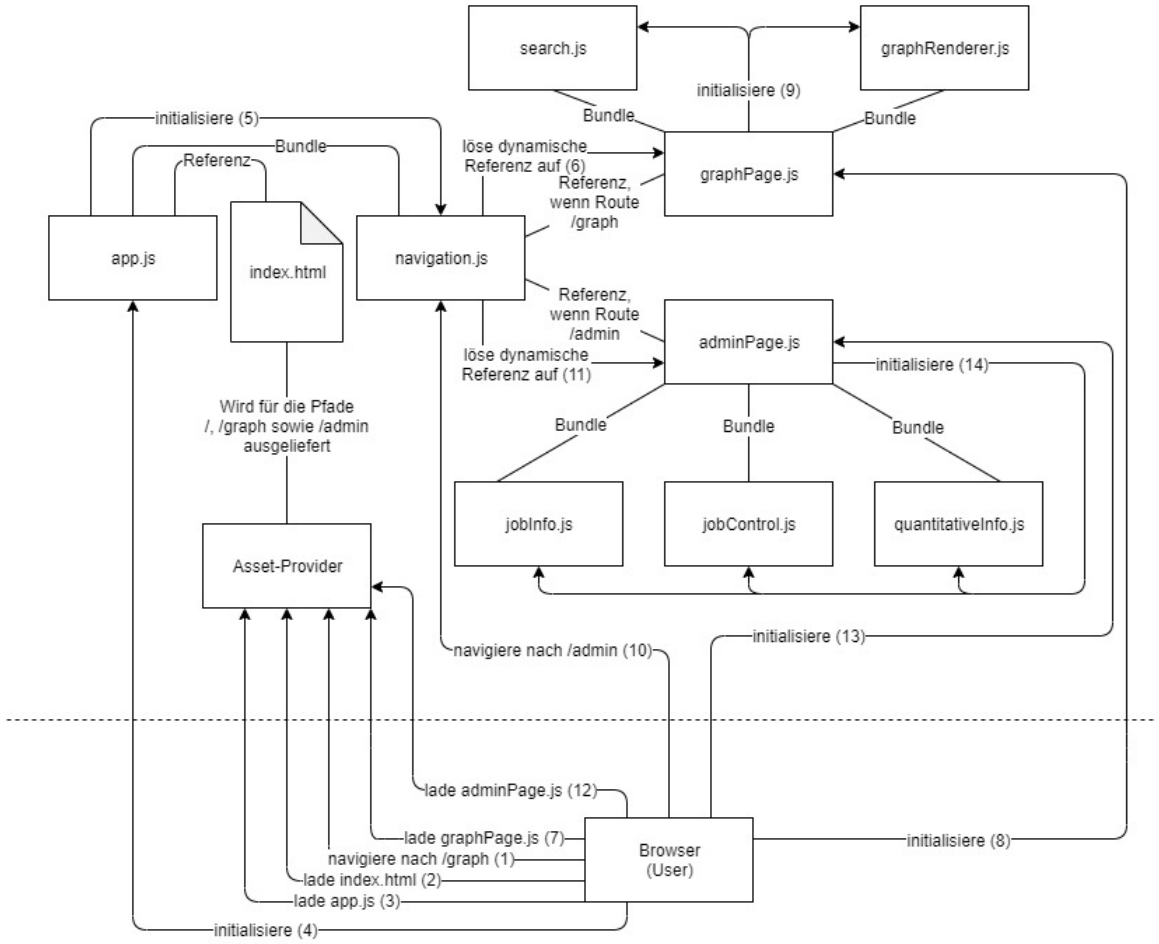

Abbildung 6: Übersicht über einen möglichen User-Flow durch das Frontend

diese werden initialisiert (9).
Jetzt navigiert der User über die Navigation auf der Seite auf die Admin-Seite. Dieser Navigationsbefehl wird, anders als der Initiale, nicht an den Asset-Provider geschickt, da jetzt das Routing über das SPA-Framework erfolgen kann (10). Die Navigation-Komponente löst durch den neuen Pfad die dynamische Referenz auf die AdminPage-Komponente auf (11) und der Browser lädt sie anschließend (12). Nun wird sie initialisiert (13), inklusive ihrer Abhängigkeiten (14).

In Abbildung 7 ist zu sehen, wie der Deployment-Prozess des Frontends funktioniert könnte. Wie zu sehen ist, wird eine Versionsnummer generiert, die für den Rest des Deployments genutzt wird (1). Diese Versionsnummer wird für das Cache-Busting sowie für ein lückenloses Deployment von Frontend und Backend benötigt. Daraufhin wird mithilfe eines Module-Bundlers ein Bundle des Frontend-Quellcodes erzeugt (2). Der Module-Bundler benutzt während des Prozesses einen Transpiler, um ein Transpilat jeder Quelldatei zu erzeugen, dass von den zu unterstützenden Browsern interpretiert werden kann (3). Diese Transpilate vereint er anschließend im Bundle, dessen Dateiname mit der eingangs generierten ID versehen wird (4). Nun wird das Bundle zum Asset-Provider hochgeladen (5). Dieser persistiert das neue Bundle, liefert aber für

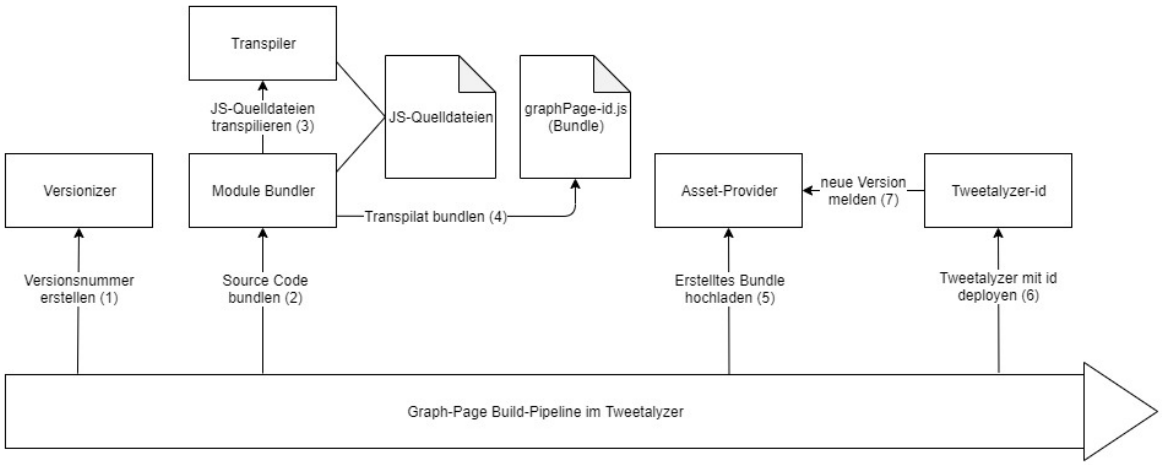

Abbildung 7: Beschreibung eines möglichen Deployment-Prozesses des Tweetalyzers

sämtliche Anfragen nach Assets nach wie vor das alte Bundle aus. Dies ist erforderlich, da es möglich ist, dass bestimmte Änderungen des Frontends gleichzeitige Änderungen des Backends benötigen. Ein Beispiel wären breaking-changes in einer Schnittstelle des Backends.

Damit das Cache-Busting funktioniert, generiert der Asset-Provider außerdem eine neue Version der index.html, inklusive sämtlicher globaler Abhängigkeiten, denn in der Navigation-Komponente muss der Link auf die entsprechende Seite, die gerade deployed wird, ausgetauscht werden.

Anschließend wird das Backend rollierend deployed, also wird die alte Instanz erst abgerissen, wenn die neu Instanz gestartet und betriebsbereit ist (6). Nachdem die neu Instanz als betriebsbereit befunden wurde, wird eine Schnittstelle des Asset-Providers aufgerufen, an der die nun aktive Versionsnummer gemeldet wird. Daraufhin liefert der Asset-Provider die neue index.html-Datei aus. Zeitgleich werden sämtliche Anfragen an den Tweetalyzer gegen die neue Instanz geleitet. Somit ist der Javascript-Cache im Browser umgangen, denn in der index.html-Datei findet sich eine Referenz auf die neue App-Komponente, inklusive Versionsnummer in der URL. Weiterhin gab es ein lückenloses Deployment von Frontend und Backend, damit zu keinem Augenblick das neue Frontend mit dem alten Backend oder umgekehrt kommunizieren könnte.

Wie zu sehen ist, ist die Architektur einer Single-Page-Application, deren Komponenten als Micro-Frontends ausgeliefert werden, recht komplex. Die konsequente Einhaltung der gleichen Architektur-Paradigmen aus dem Backend bietet allerdings auch die gleichen Vorteile, wie unabhängige und schnellere Deployments, Skalierbarkeit, auch über potentielle Teamgrenzen hinweg, sowie einen vereinheitlichten Asset-Delivery-Prozess. Ein weiterer Vorteil ist weitestgehende technologische Unabhängigkeit. Da die verschiedenen Frontent-Komponenten aus unterschiedlichen Microservices geliefert werden, können sowohl für den Quellcode als auch für den Build-Prozess genau die Tech-

nologien genutzt werden, die die speziellen Anforderungen an ebenjene Komponente am besten erfüllt.

6 Technische Realisierung

Dieses Kapitel widmet sich der Beschreibung der technischen Realisierbarkeit der gewünschten Funktionen der Software. Hierbei wird auf Grundlage der beschriebenen Architektur und unter Berücksichtigung der gewünschten Features pro identifizierter Funktionalität in Unterabschnitt 4.1 eine detaillierte Beschreibung einer möglichen Implementierungsstrategie aufgezeigt. Da es sich bei der zu konzipierenden Software um ein Expertentool handelt, ist zunächst kein User-Management berücksichtigt worden. Soll diese Applikation aber mehrere Benutzer gleichzeitig versorgen oder öffentlich aus dem Internet erreichbar sein, muss aus Sicherheitsgründen zwangsläufig ein User-Management, inkl. einer Absicherung der Backend-Schnittstellen mit einem Authentifizierungs- sowie Autorisierungsmechanismus, implementiert werden.

6.1 Hochladen der Daten

Um das Feature F1, Daten-Upload, erfüllen zu können, ist es erforderlich, dass es im Frontend ein Datei-Upload-Formular gibt, mit dem ein User die hochzuladene Datei auswählen kann. Weiterhin muss es eine Schnittstelle im Backend geben, die den gewünschten Datensatz in Empfang nimmt. Da es sich um einen großen Datensatz handeln kann, ist es sinnvoll, ein geeignetes Dateiformat vorzugeben. Das bereits in dem Datensatz, der dieser Arbeit zu Grunde liegt, gewählte Datenformat, eignet sich hierfür hervorragend. Insofern akzeptiert die Schnittstelle ein Zip-Archiv, das wiederum eine Liste an Archiven von JSON-Dateien im G-Zip-Format, wie in Unterabschnitt 2.3 beschrieben, enthält.
Nachdem die Datei in der Schnittstelle in Empfang genommen wurde, sollte sie von der Applikation persistiert werden. Als Speicherort würde sich beispielsweise ideal AWS S3 oder eine vergleichbare Speicher-Lösung anbieten. Die Anforderungen sind nicht besonders hoch, es ist keine extreme Performance notwendig, da es sich beim Speichern sowie beim Lesen nicht um zeitkritische Operationen handelt. Weiterhin sollte der Speicherort selbstverständlich in der Größe skalierbar sein, dies ist aber mittlerweile der defakto-Standard und somit keine wirkliche Anforderung mehr.
Nach dem Speichern sollte das Frontend eine Antwort aus dem Backend erhalten, die den Zustand des Datei-Uploads mitteilt. Dies sollte über den Http-Response-Code vermittelt werden. Der Response-Code sollte daraufhin ausgewertet und das Resultat dem User verständlich angezeigt werden.
Da der User potentiell sehr große Dateien hochladen kann, gilt es, eine Reihe von Problemen zu lösen. Als erstes ist es keine gute Lösung, die Datei als Ganzes in Empfang zu nehmen, bevor diese von der Anwendung persistiert wird. Dies würde es erfordern, die gesamte Datei zu einem Zeitpunkt im Arbeitsspeicher zu halten, was neben Performance-Problemen auch ein großes Security-Risiko darstellt, da es so für einen

potentiellen Angreifer ohne viel Aufwand möglich ist, den kompletten Arbeitsspeicher der Applikation zu füllen und diese somit zum Absturz zu bringen. Um dieses Problem zu lösen, bietet sich ein Daten-Streaming mithilfe des http2-Multiplexings an [Gri13]. Durch das Multiplexing können Request-Payloads in mehreren Teilen anstelle der Gesamtdatei geschickt werden. Diese Einzelteile sollten dann von der Applikation in Empfang genommen und, anstelle sie im Arbeitsspeicher zu halten, direkt an das Speichermedium weitergeschickt werden. Würde hierfür beispielsweise AWS S3 genutzt, böte sich der Multipart-Upload an [AWS20]. Hierdurch kann der Arbeitsspeicher, in dem der jeweilige Teil der Gesamtdatei, die gerade von der Applikation angenommen wurde, sofort wieder freigegeben werden, wenn der jeweilige Teil zu S3 hochgeladen ist. Ein weiteres Problem ist, dass der User abhängig von der Dateigröße, seiner Internetverbindung und der Upload-Geschwindigkeit der Applikation Richtung persistenter Speicher mitunter sehr lange warten muss, bis der Upload abgeschlossen ist. Um dies zu verdeutlichen, ein kurzes Beispiel, bei dem von dem utopischen Szenario ausgegangen wird, dass der Upload zum persistenten Speicher keine Zeit kostet. Daher würde es in der Praxis zu einem noch schlechteren Ergebnis kommen: Der User will einen Datensatz mit einer Größe von 800 Megabyte hochladen. Dies entspricht ca. der Größe des vorliegenden Datensatzes. Ein mittelmäßiger VDSL-Anschluss besitzt ein Upload-Limit von 40 Megabit pro Sekunde, von denen aufgrund von Parallelnutzung in der Regel nicht mehr als 75% für den konkreten Anwendungsfall zur Verfügung stehen [Tel20]. Dies würde, wenn die Geschwindigkeit wie angegeben genutzt werden könnte, einen Upload von 3,5-4 Minuten bedeuten.
Da auch noch größere Datensätze anfallen könnten und schon diese Dauer sehr lang ist, ist ein alternatives Upload-Szenario denkbar. Hierbei müsste der User die Datei zunächst öffentlich verfügbar machen, beispielsweise, weil diese bereits auf einem Server liegt, der aus dem Internet erreichbar ist. Da diese Einschränkung definitiv nicht alle Anwendungsszenarien abdeckt, ist sie nur als Ergänzung zur bereits genannten Methode zu verstehen. Nun könnte der User die URL, unter der die Datei verfübar ist, anstelle der Datei direkt im Frontend angeben. Diese Information würde dann an das Backend geschickt werden, wo anschließend ein asynchroner Prozess angestoßen wird, der die Datei herunterlädt und auf bereits beschriebenem Wege persitiert. Da dieser Prozess asynchron ist, muss ein Monitoring des aktuellen Status dieses asynchronen Prozesses für den User zur Verfügung gestellt werden, da dieser sonst keine Chance hat, eine Rückmeldung über den Datei-Upload zu erhalten.

6.2 Quantitative Aspekte des Datensatzes

Um das Feature F2 bereitzustellen, muss es im Frontend einen Abschnitt geben, in dem die gewünschten Daten angezeigt werden können. Hierbei sollten verschiedene Darstellungsformate für die unterschiedlichen quantitativen Aspekte genutzt werden,

je nachdem, welche Darstellungsart angemessen ist. Verschiedene Anzahlen oder andere Kennzahlen können beispielsweise als einfache Zahl angegeben werden. Da es sich bei allen identifizierten quantitativen Aspekten um solche Kennzahlen handelt, können alle dieser Kennzahlen als einfache Zahl visualisiert werden. Gäbe es beispielweise Anforderungen mit temporalem Bezug, würde sich ein 2D-Graph anbieten, für Verhältnisse ein Tortendiagramm. Die konkreten Anforderungen an das Frontend sind also abhängig von der jeweiligen KPI, die visualisiert werden soll. Daher sollte ein flexibles Layout gewählt werden, das diese verschiedenen Aspekte bieten kann.
Die eigentlichen Daten für die Visualisierung müssen aus dem Backend geliefert werden. Dafür sollte eine Schnittstelle pro KPI implementiert werden, die jeweils die geforderte KPI liefert. Da sich die gewünschten KPIs sowohl auf die Rohdaten als auch auf die Graphdaten beziehen, muss für die Ermittlung der jeweiligen KPI die korrekte Datenbank genutzt werden. Sämtliche gewünschten KPIs können durch simple Datenbank-Abfragen ermittelt werden. Sollte eine KPI gewünscht sein, die sich nicht so leicht ermitteln lässt oder deren Ermittlung einfach viel Zeit in Anspruch nimmt, können diese Ermittlungen auch periodisch stattfinden und das Ergebnis anschließend in der Applikation gecached werden.

6.3 Zurücksetzen der Applikation

Für die Implementierung des Features F3 ist ein Button im Frontend erforderlich, der den gewünschten Request an das Backend schickt. Da es sich hierbei um einen kompletten Daten-Reset handelt, muss der Knopf so abschreckend gekennzeichnet werden, dass ein User diesen nicht versehentlich klickt, beispielsweise durch roten Text. Außerdem sollte der User den Button-Klick in einem Layer oder Ähnlichem bestätigen müssen.
Im Backend ist wieder eine Schnittstelle erforderlich, die auf den Request reagiert und daraufhin einen Löschbefehl an den persistierten Rohdatensatz sowie an sämtliche Datenbanken sendet. Sind alle Daten gelöscht, wird dieses Ergebnis über den entsprechenden Http-Response-Code an das Frontend übermittelt, wo ein entsprechender Hinweis angezeigt werden kann.

6.4 Steuerung/ Administration der Plattform

Um Feature F4 sowie F5 zu implementieren, ist es unbedingt notwendig, eine Möglichkeit zu schaffen, asynchrone Prozesse im Backend ausführbar zu machen. Da im Rahmen des Datenimport- und Verarbeitungsprozesses eine ganze Reihe an asynchronen Funktionalitäten benötigt werden, sollte dies als generische Komponente implementiert werden. Dies birgt den Vorteil, dass es auch in Zukunft leicht möglich ist, Veränderungen oder Ergänzungen an den konkreten asynchronen Prozessen vorzunehmen, ohne Einfluss auf die Prozesse zu nehmen und vor allem, ohne den generischen Teil jedes mal neu zu implementieren.

Die asynchronen Prozesse lassen sich gut als Jobs beschreiben. Jeder Job hat einen Start- und Endzeitpunkt, einen Status sowie die Möglichkeit, Auskunft über den aktuellen Fortschritt zu geben. Weiterhin besteht die Möglichkeit, dass gewisse asynchrone Prozesse nicht parallel zu einander laufen dürfen, da diese sonst Seiteneffekte auf einander haben würden. Um dies zu verhindern sollte jeder Job eine Mutex-Gruppe mit anderen Jobs angeben, zu denen er nicht parallel ausgeführt werden darf. Im Folgenden wird eine Beschreibung des Frameworks gegeben, das entwickelt werden muss, um F4 und F5 mit den gerade genannten Rahmenbedingungen bereitzustellen.
Im Frontend muss es die Möglichkeit geben, eine Auflistung aller Jobs einzusehen. Ferner muss jeder Job in dieser Auflistung Informationen zu Fortschritt und Zustand des Jobs beinhalten. Um einen Job zu starten soll ein Button implementiert werden, der auf Klick einen Request ins Backend sendet, ebenjenen Job zu starten.
Im Backend muss ein Service implementiert werden, der für die Jobsteuerung verantwortlich ist. Er sollte eine Referenz auf alle registrierten Jobs und ihre Zustände haben. Weiterhin sollte er die Möglichkeiten bieten, einen Job zu starten sowie Auskunft über den Zustand eines Jobs zu geben. Dieser Service sollte ferner die volle Verantwortung darüber haben, ob ein Job ausgeführt werden darf oder nicht. Beispielsweise sollte er eine Jobausführung verhindern, wenn der Job aktuell bereits ausgeführt wird oder sich der auszuführende Job in einer Mutex-Gruppe eines bereits laufenden Jobs befindet.
Weiterhin wird eine Definition eines Jobs benötigt. Eine Möglichkeit, einen Job zu definieren, wäre über eine abstrakte Basisklasse. Diese Klasse würde dann generische Logik wie die benötigten Felder, Exception-Handling, Zeit- sowie Fortschrittsmessung beinhalten. Über eine abstrakte Methode kann der genaue Zeitpunkt in der Ausführung des Jobs angegeben werden, an der die eigentliche Business-Logik eines konkreten Jobs ausgeführt werden soll.
Neben dem Service bedarf es einer Schnittstelle, die den Wunsch, einen Job zu starten, entgegennimmt, und mithilfe des beschriebenen Job-Services versucht, ebenjenen Job zu starten. Anschließend sollte das Ergebnis dieses Versuchs als Response-Code zurückgegeben werden. Da die Jobs asynchron ausgeführt werden, bedeutet ein erfolgreicher Response-Code an dieser Stelle lediglich, dass der Job gestartet wurde und gibt in keinster Weise Aufschluss über das Ergebnis des eigentlichen Jobs.
Es wird eine weitere Schnittstelle benötigt, die Auskunft über den Zustand eines geforderten Jobs gibt. Dafür sollten die für das Frontend benötigten Informationen über Zustand und Fortschritt zurückgegeben werden.
Der beschriebene Ansatz hat das Problem, dass die aktuellen Zustände der Jobs nur im Arbeitsspeicher der laufenden Anwendung vorliegen. Sollten beispielsweise mehrere Instanzen der Anwendung parallel ausgeführt werden, kommt es zu Synchronisationsproblemen zwischen den einzelnen Instanzen. Daher müsste der Zustand eines Jobs über eine Datenbank an alle Instanzen verteilt werden. Somit würden die Instanzen wieder synchronisiert sein.

6.5 Suchmaschine

Für das Feature F6 bietet es sich im Frontend an, für die verschiedenen Filtermöglichkeiten jeweils ein Eingabefeld für das entsprechende Filterkriterium bereitzustellen. Weiterhin sollte es einen Button geben, über den die Filteranfrage an die Anwendung geschickt werden kann. Da es sich beim Feature F6 primär um einen Einstiegspunkt in das Feature F7 handelt, ist es sinnvoll, diese beiden Komponenten im Frontend zu verknüpfen.
Im Backend werden innerhalb des Repositories, das den Zugriff auf die Graphdatenbank ermöglicht, 3 Methoden benötigt, die jeweils andere Queries auf der Datenbank ausführen. Die erste Methode nimmt einen Usernamen entgegen und durchsucht hiermit die Graphdatenbank nach User-Knotenpunkten, die ebenjenen Usernamen enthalten. Da ein Username eindeutig ist, kann maximal ein Knotenpunkt gefunden werden. Ausgehend von einem gesetzten Limit, das die Anzahl an gewünschten Knotenpunkten begrenzt, sowie einer Iterationstiefe, die angibt, wie weit vom ursprünglichen Knotenpunkt maximal am Graph traversiert werden darf, werden dann die entsprechenden Knotenpunkte entlang der Relationen des ursprünglichen User-Knotenpunktes aus der Graphdatenbank geladen.
Die Methode 2 soll einen Hashtag entgegennehmen und daraufhin Tweets identifizieren, die diesen Hashtag enthalten. Ausgehend von diesen Tweets wird unter dem gleichen Prinzip mithilfe des Limits und der Iterationstiefe weitere Knotenpunkte entlang des Graphen identifiziert, die mit ausgeliefert werden.
Die dritte Methode kombiniert diese beiden Ansätze nun. Sie sucht mithilfe eines Usernamens und eines Hashtags nach dem User mit dem angegebenen Namen, der weiterhin eine Relation zu Tweets mit dem angegebenen Hashtag hat.
Diese drei Methoden werden nun in einer Schnittstelle zugänglich gemacht. Diese akzeptiert eine beliebige Kombination an Filterkriterien als Parameter, sowie ein Limit und die Iterationstiefe.

6.6 Darstellung des Datensatzes als interaktives Netzwerk

Um das Feature F7 umzusetzen, muss im Frontend eine Konponente implementiert werden, die einen Graphen rendern kann. Diese soll die bereits bestehende Schnittstelle für die Suche im Backend nutzen, um die Daten zu beziehen, aus denen der Graph gerendert werden soll. Da es auch möglich ist, der Schnittstelle keinen Filterparameter zu übergeben, erfüllt sie die Anforderungen, um die Daten für den Graphen bereitzustellen.
Zunächst stellt sich die grundlegende Frage des Layoutings. Wie in Unterabschnitt 3.3 bereits beschrieben, bietet sich ein Layouting, das auf einer Kräfte-Simulation basiert, an.
Sind die Daten geladen, muss das Frontend diese rendern. Es stellt sich die Frage, ob

der Graph als Vektorgrafik (SVG) oder als html-canvas gerendert werden soll. Beide Methoden haben unterschiedliche Vor- und Nachteile. So ist das Rendern von vielen Elementen deutlich performanter mit canvas [Smu09]. SVG hat allerdings den Vorteil, dass Interaktivität deutlich einfacher zu implementieren ist, da der Browser den Canvas-Bereich nur als ein Element kennt und die Komponenten einer SVG-Grafik als komplette Hierarchie für den Browser bekannt sind [Cru11]. Da es eine Kernanforderung ist, eine interaktive Lösung zu entwickeln, sollte also SVG als Rendering-Grundlage verwendet werden.
Elemente in einer SVG-Grafik haben jeweils eine X-und Y-Koordinate. Anhand derer wird ihre Position in der Grafik festgelegt. Da die Daten aus dem Backend keine Information über ihre Position im Layout haben, muss diese über ein Layouting-Prozess berechnet werden. Da dieser auf einer Kräfte-Simulation basieren soll, muss so eine Simulation implementiert werden. Da diese Kräfte-Simulation sehr komplex ist, sollte hier auf eine bestehende Bibliothek wie zum Beispiel d3-force [d320] zurückgegriffen werden.
Initial werden den Knotenpunkten zufällige Startpositionen zugewiesen. Nun wird diese Simulation für eine begrenzte Zeit ausgeführt. Mit jedem Simulationsschritt werden die X-und Y-Koordinaten der einzelnen Knotenpunkte aufgrund der Ergebnisse der Anwendung Kräfte-Funktionen verändert. Somit wird iterativ ein Layout erzeugt, das durch die durch die registrierten Kräfte vorgegebenen Rahmenbedingungen definiert ist.

6.6.1 Begrenzung der Knotenpunkte

Um die anzuzeigenden Knotenpunkte zu begrenzen, so wie in F7.1 beschrieben, soll ein Eingabefeld implementiert werden, in das das gewünschte Limit eingegeben werden kann. Dieses Eingabefeld sollte aus Konsistenzgründen in das Formular mit den Suchkriterien integriert werden, sodass sämtliche Konfigurationsmöglichkeiten an einer Stelle zu finden sind. Das gewünschte Limit wird dann dem Backend übergeben und wie bereits beschrieben verwendet, um das Suchergebnis zu begrenzen.

6.6.2 Einflussnahme auf das Rendering

In F7.2 ist die Anforderung gestellt worden, dass ein User Einfluss auf das Layout des Graphen nehmen können soll. Zu diesem Zweck sollen für die unterschiedlichen Parameter, die die Berechnungsgrundlage für das Layout sind, Eingabefelder implementiert werden. Über diese Werte kann der User dann Einfluss auf das Layout nehmen. Weiterhin soll es möglich sein, einzelne Knotenpunkte im gerenderten Graphen via Drag&Drop zu verschieben. Hierfür muss für das Drag-Event ein Javascript-Listener implementiert werden, der dann die X-und Y-Koordinaten des zu verschiebenden Knotenpunktes immer auf die X-und Y-Koordinaten des Mauszeigers setzt. Während des

Verschiebens sollte die Kräfte-Simulation des Layoutings wieder ausgeführt werden, damit das Layout wie aus einem Guss wirkt.

6.6.3 Kontextsensitivität

Die verschiedenen Knotenpunkte haben eine Reihe von Attributen, die im Graphen angezeigt werden sollen, beispielsweise der Text eines Tweets oder der Name eines Users. Wie in Unterunterabschnitt 3.3.5 beschrieben und in F7.3 gefordert, sollen diese Informationen nur angezeigt werden, wenn der User mit der Maus über einen Knotenpunkt fährt. Die Informationen sollten in einem Overlay über den Graphen neben dem Knotenpunkt, über dem gerade die Maus steht, angezeigt werden. Ferner soll der Graph ausgegraut werden, mit Ausnahme von dem aktuellen Knotenpunkt, allen Knotenpunkten, die über eine direkte Relation mit ebenjenem Knotenpunkt verbunden sind sowie den Relationen selbst.
Um dies zu ermöglichen, muss zunächst beim Rendering der SVG-Grafik ein so beschriebenes Overlay für jeden Knotenpunkt mit in den Graphen gerendert werden. Dies wird wie beschrieben positioniert und mit den entsprechenden Informationen befüllt. Dann wird es über ein Styling unsichtbar gemacht. Weiterhin muss ein Event-Listener auf das MouseOver-Event gelegt werden, der dann den Graphen wie beschrieben ausgraut und das entsprechende Overlay sichtbar macht.

6.6.4 Unterscheidbarkeit der Knotenpunkte

Um die Knotenpunkte analog zu F7.4 besser unterscheidbar zu machen sollen sich diese in Größe und Farbe unterscheiden. Dabei soll der Typ des Knotenpunktes durch die Farbe und die Anzahl an Kanten sowie durch die Größe des Knotenpunktes repräsentiert werden. Beide Attribute, Farbe und Größe, können während des Rendering-Prozesses der SVG-Grafik gesetzt werden. Weiterhin können nach gleichem Prinzip auch die Kanten eingefärbt werden. Somit kann der Typ der Kante durch die Farbe repräsentiert werden. Auf diesem Wege müssen keinerlei Label mehr am Graphen angezeigt werden, da relevante Daten durch die Hover-Interaktion eingesehen werden können und Informationen, die einen Überblick über den Gesamtgraphen bieten, durch natürliche Attribute der Elemente des Graphen, Farbe und Größe, gegeben werden können.

6.6.5 Navigation durch das Netzwerk

Damit sich ein User wie in F7.5 gefordert durch das Netzwerk bewegen kann, soll es möglich sein, den Bildausschnitt durch Zoomen und Verschieben zu verändern. Für das Zoomen kann dies durch ein Skalieren der Elemente mithilfe des SVG-Attributes Viewbox realisiert werden. Dabei muss beachtet werden, dass sämtliche Texte im Graphen nicht skaliert werden dürfen. Da die Viewbox aber pauschal die gesamte Grafik skaliert, muss entsprechend an den Textelementen gegenskaliert werden, damit die

Viewbox-Skalierung ausgeglichen wird. Das Verschieben des Bildausschnittes lässt sich ebenfalls mithilfe der SVG-Viewbox realisieren.

6.7 Datenimport- und Verarbeitungsprozess

Die Grundlage für die beschriebenen Features ist der Datenimport des Zip-Archives. Dieser lässt sich in 2 Prozessschritte aufteilen: Rohdatenimport und Graphimport. Eine Aufteilung in diese Prozessschritte ist sinnvoll, denn diese unterscheiden sich stark in Laufzeit und Zweck. Weiterhin soll der Datenimportsprozess leicht erweiterbar sein. Sollen die Daten beispielsweise bereinigt, angereichert, gefiltert, sortiert oder aggregiert werden, bevor diese in die Graphdatenbank importiert werden, kann dies auf Grundlage der importierten Rohdaten passieren. Somit führt auch eine nachträgliche Erweiterung der Software nicht zu einem Neuimport des Zip-Archives. Insofern ist die Abhängigkeit zu der eigentlichen Rohdatendatei minimiert.
Ein weiterer Grund, der den Rohdatenimport rechtfertigt, ist, dass es sonst deutlich komplexer wäre, eine Datengrundlage auf Basis mehrerer Rohdatendateien zu erzeugen. Soll die Datengrundlage beispielsweise nach und nach um weitere Daten erweitert werden, ist dies mit dem beschriebenen Prozess problemlos möglich, da die Rohdaten in einer Datenbank gespeichert werden, die beispielsweise Eindeutigkeit über einen Index sicherstellt.
Noch ein Grund für diese Aufteilung ist, dass einige Analysen deutlich leichter auf einer Mongo-DB durchgeführt werden können. Durch das Aggregation-Framework der Mongo-DB sind komplexe Analysen wie Map-Reduce-Abfragen möglich. Durch eine in verschiedene Schritte aufgeteilte Aggregation-Pipeline können beispielsweise durch klug gewählte Filter extreme Performance-Gewinne im Vergleich zu Cypher-Queries mit neo4j gewonnen werden.
In Abbildung 8 ist der Datenimportprozess auf einer abstrakten Ebene visualisiert.

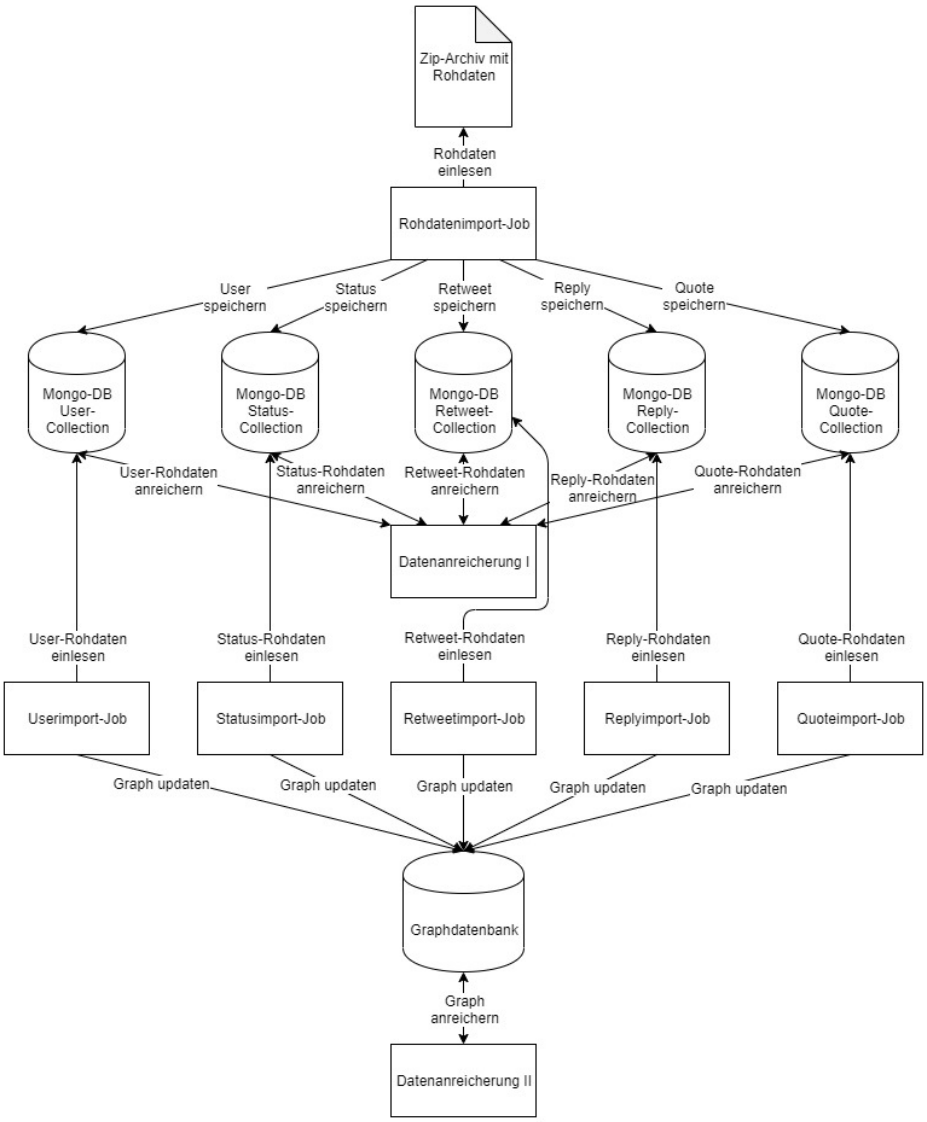

Abbildung 8: Visualisierung des gesamten Datenimport-Prozesses

6.7.1 Rohdatenimport

In Abbildung 9 ist der Rohdatenimport zusammengefasst dargestellt. Wie zu sehen ist, handelt es sich um einen Job im bereits beschriebenen Job-Framework. Der Job wird durch den User über das Frontend ausgelöst. Einmal gestartet, liest er zunächst das Zip-Archiv. Daraufhin wird es komplett entpackt und die einzelnen Dateien, die Rohdaten im JSON-Format enthalten, werden in die Mongo-DB importiert. Dabei wird über das Attribut type am Rohdatensatz entschieden, in welche Collection der Datensatz importiert werden soll. Das Handling mit dem Zip-Archiv soll hierbei durch einen eigenen Service übernommen werden. Dieser Service sollte einen kontinuierlichen Datenstrom an Rohdaten liefern, der dann von einem weiteren Service, der den eigentlichen Import in die Datenbank steuert, genutzt wird. Somit ist der Zugriff auf die Rohdatendatei komplett gekapselt und kann in der Zukunft einfach auf einen anderen Mechanismus, beispielsweise den direkten Konsum der Twitter-API, umgestellt werden.

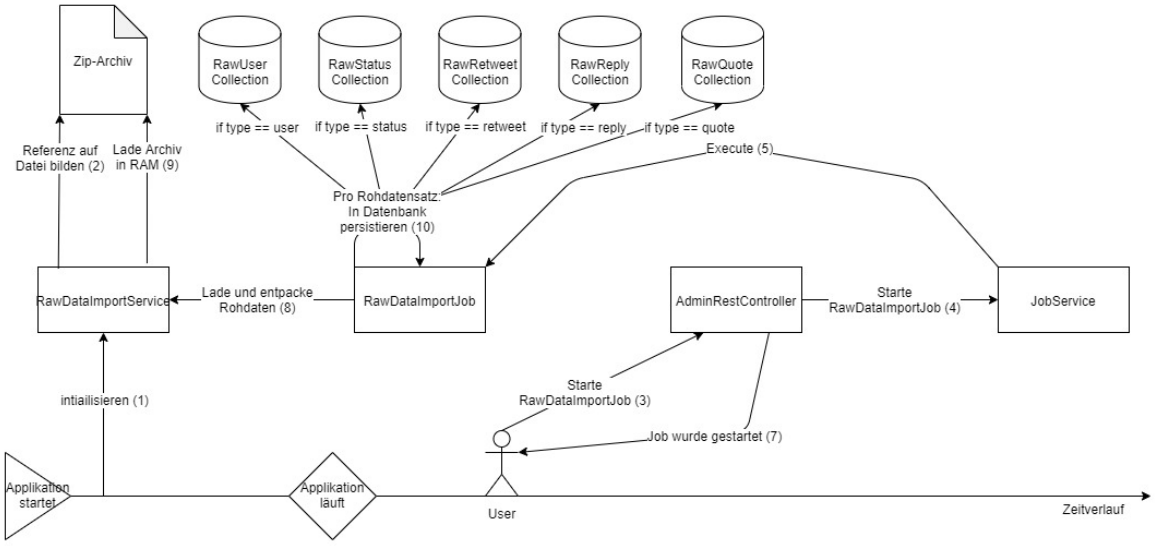

Abbildung 9: Visualisierung des Rohdatenimport-Prozesses

Sollten die Rohdaten in irgendeiner Form bereinigt werden, wäre dies der ideale Zeitpunkt. Ein Datensatz in der MongoDB sollte mit einem bereinigten Rohdatensatz gleichgesetzt werden, um die Komplexität für die nachgelagerten Prozesse so gering wie möglich zu halten.

6.7.2 Datenanreicherung Teil 1

Sollen die Rohdaten angereichert werden, sollte dies idealerweise geschehen, bevor diese in die Graphdatenbank importiert werden, da ein Update in der Mongo-DB deutlich performanter ist, da es zwischen den Datensätzen keine Beziehungen auf Datenbankebene gibt. Für die geforderten Funktionalitäten ist zwar keine Datenanreicherung notwendig, vor dem Hintergrund, dass die konzipierte Software aber als generelle Grundlage für die Analyse von Twitter-Daten dienen soll, sollte auf diese Erweiterbarkeit schon bei der grundlegenden Implementierung geachtet werden.
Eine Anreicherung sollte ebenfalls durch asynchrone Jobs durchgeführt werden. Das notwendige Framework dafür steht bereit.

6.7.3 Import in die Graphdatenbank

Nachdem die Daten normalisiert sowie angereichert wurden, können diese in die Graphdatenbank importiert werden. Auch dies wird wieder durch Jobs durchgeführt. Da die verschiedenen Typen von Rohdaten in unterschiedlichen Mongo-DB-Collections vorliegen und eine unterschiedliche Logik beim Import benötigen, sollte für jeden dieser Typen ein Job angelegt werden, der die jeweiligen Rohdaten aus der Mongo-DB liest und diese in die Graphdatenbank importiert.
Da Graphdatenbanken mit Relationen und Knotenpunkten arbeiten, müssen die Rohdaten, die in Dokumentenform vorliegen, in diese Form gebracht werden. Hierfür wird

das in Abbildung 1 beschriebene Schema genutzt. Es gibt also 2 Knotenpunkte, User und Tweets. Zwischen diesen Knotenpunkten existieren nun die dargestellten Relationen. Somit sind sämtliche Interaktionen zwischen den verschiedenen Entitäten im Twitter-Kontext abbildbar.
Um jetzt beispielsweise einen User sowie einen Tweet, der von ebenjenem User getweetet wurde, zu importieren, müssen die folgenden Schritte ausgeführt werden:

- Der User muss als Knotenpunkt in die Graphdatenbank aufgenommen werden
- Der Tweet muss als Knotenpunkt in die Graphdatenbank aufgenommen werden
- Sind User und Tweet in der Graphdatenbank, muss zwischen diesen beiden Knotenpunkten eine Relation vom Typ tweets erstellt werden

Um eine Relation zu erstellen, müssen also erst beide Knotenpunkte in der Graphdatenbank erstellt sein. Aus dieser Bedingung ergeben sich dann die folgenden Jobs:

- Userimport-Job, beschrieben in Abbildung 10
- Statusimport-Job, beschrieben in Abbildung 11
- Replyimport-Job, beschrieben in Abbildung 12
- Retweetimport-Job, beschrieben in Abbildung 14
- Quoteimport-Job, beschrieben in Abbildung 13

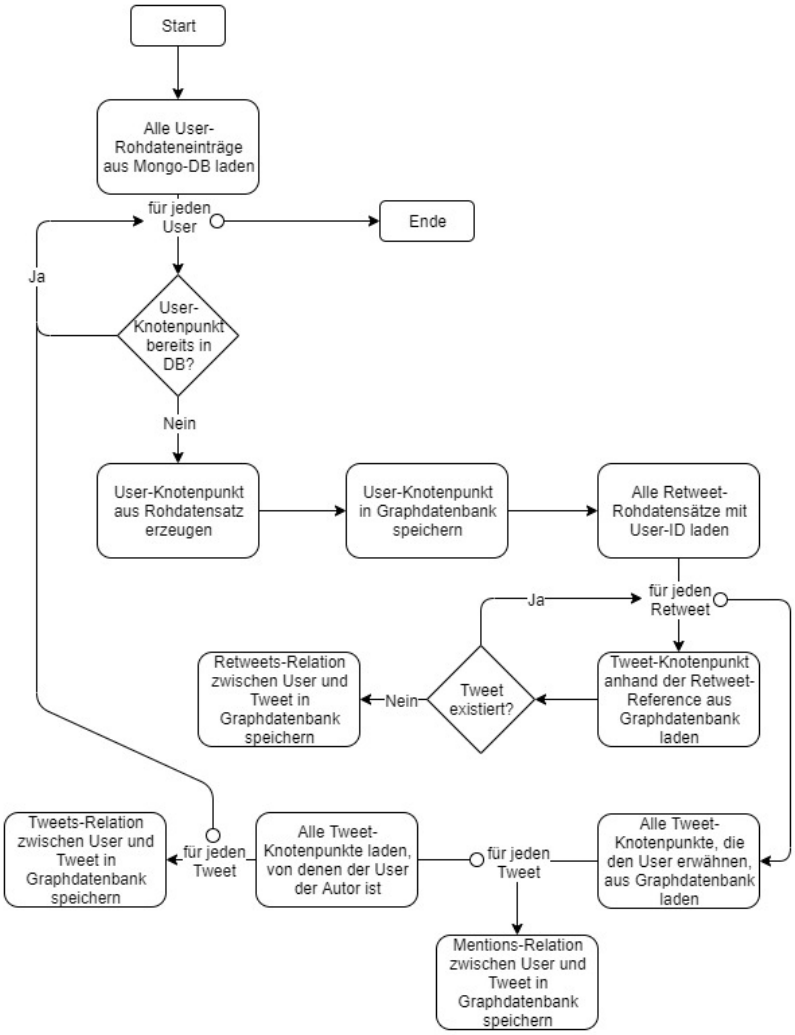

Abbildung 10: Visualisierung des User-Imports in die Graphdatenbank

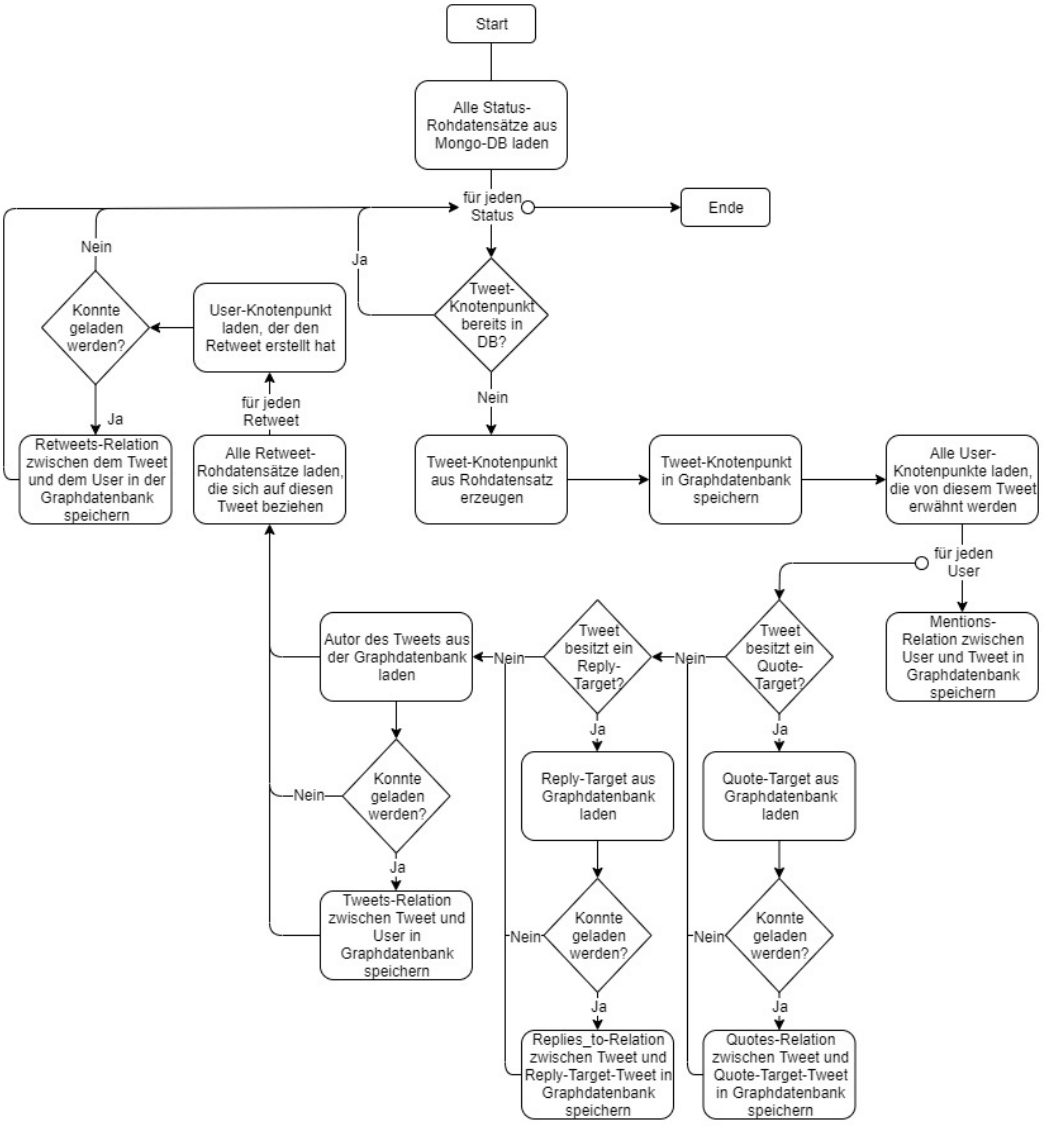

Abbildung 11: Visualisierung des Status-Imports in die Graphdatenbank

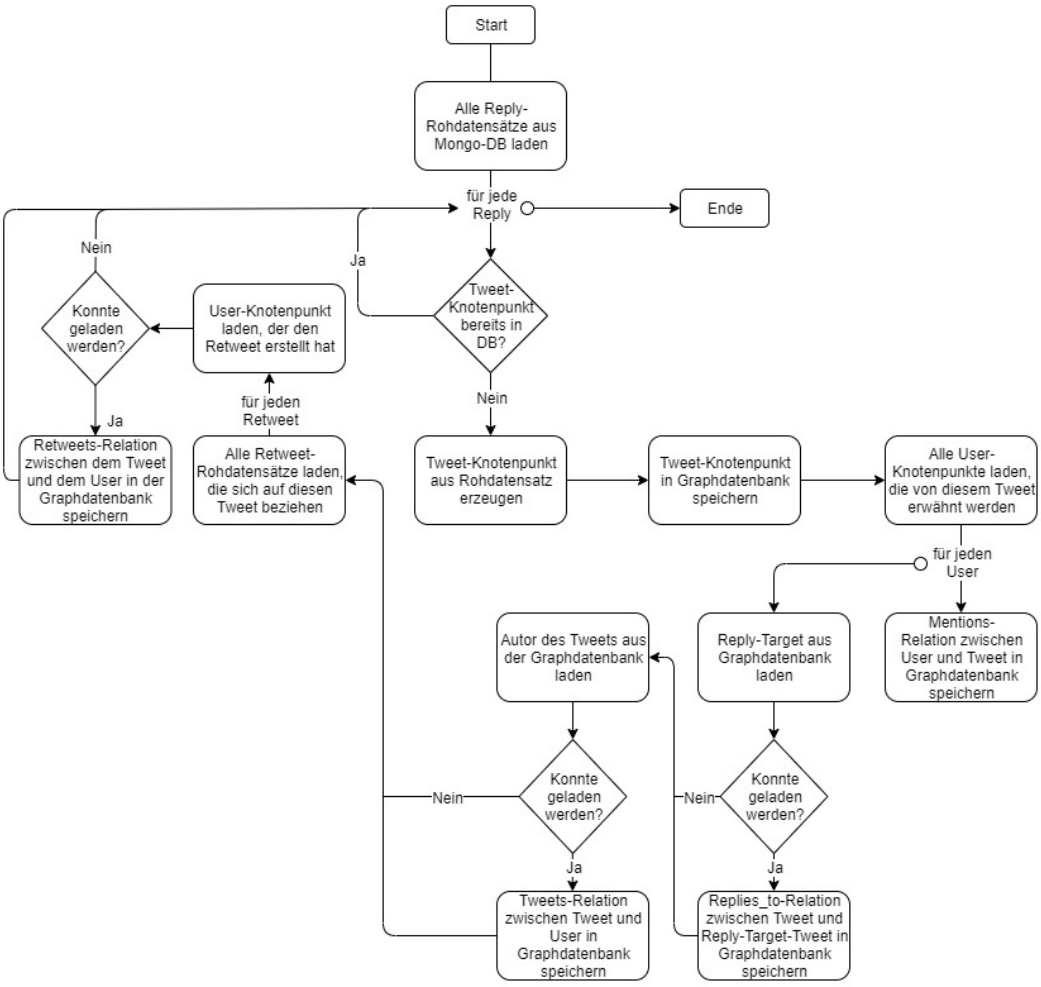

Abbildung 12: Visualisierung des Reply-Imports in die Graphdatenbank

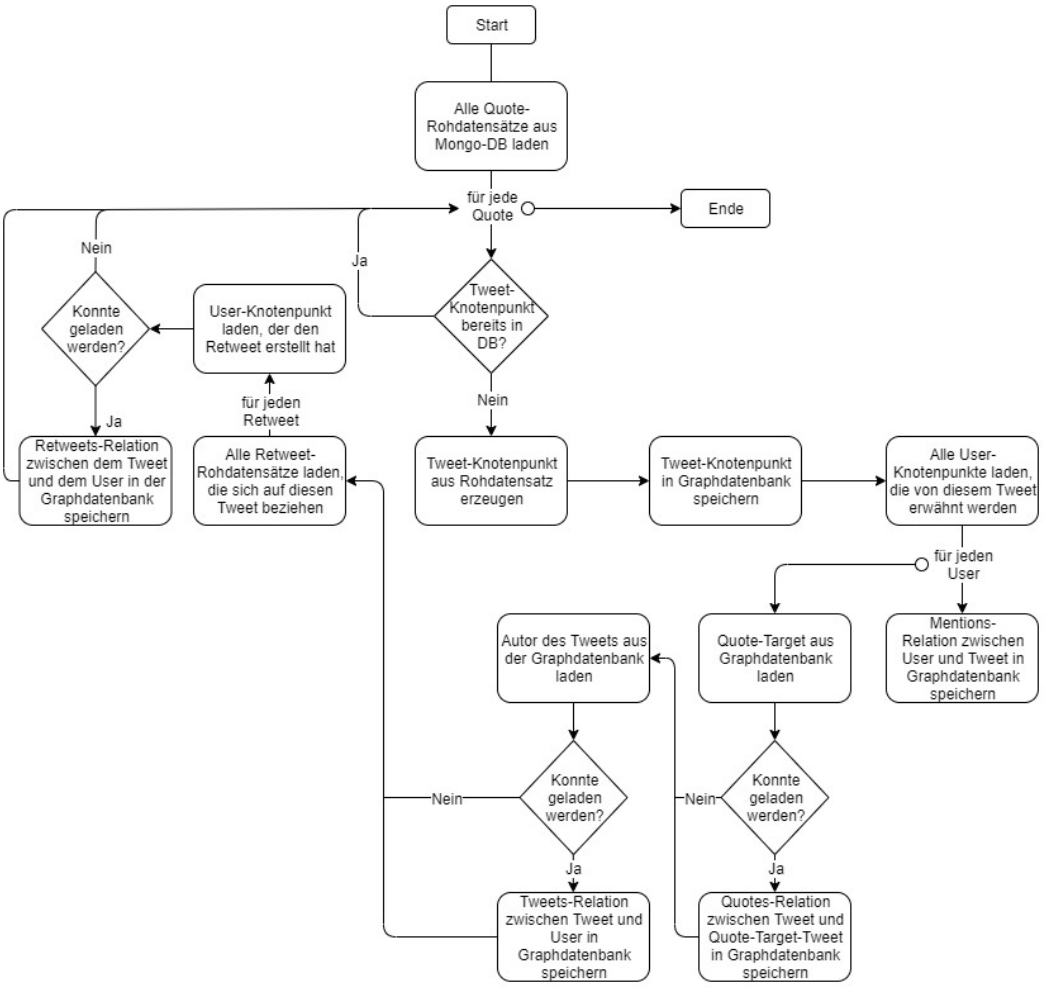

Abbildung 13: Visualisierung des Quote-Imports in die Graphdatenbank

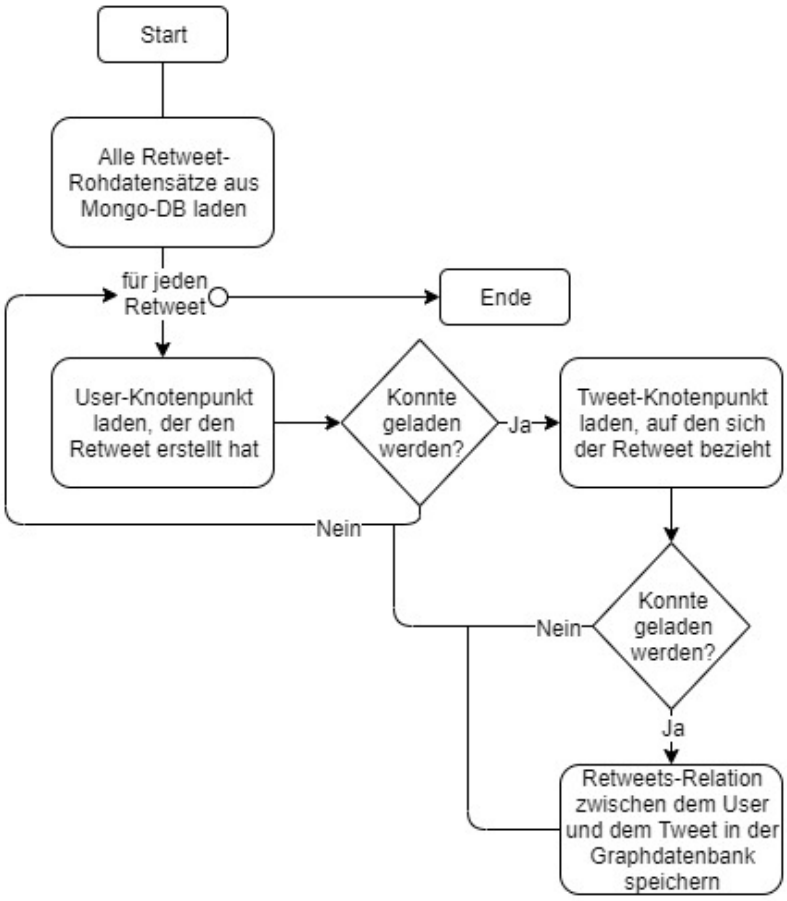

Abbildung 14: Visualisierung des Retweet-Imports in die Graphdatenbank

6.7.4 Datenanreicherung Teil 2

Wenn für bestimmte Anreicherungsprozesse die Graphdatenbank notwendig ist, können diese nun durchgeführt werden. Anschließend werden die Daten in der Graphdatenbank als freigegeben markiert und können ab diesem Zeitpunkt abgerufen werden.

7 Implementierung des Prototypen

Dieses Kapitel widmet sich der Beschreibung des implementierten Prototypen. Zunächst wird aufgezeigt, welche fachlichen Features implementiert wurden. Daraufhin wird aufgezeigt, welche Kompromisse aufgrund der prototypischen Natur der Software im Vergleich zur geplanten Architektur gemacht werden mussten. Anschließend werden einige besonders komplexe Teile der Implementierung erklärt. Abschließend findet eine rückblickende Betrachtung und Bewertung des Implementierungsprozesses statt.

7.1 Implementierte Features

In Tabelle 2 ist aufgeführt, welche Features im Rahmen des in dieser Arbeit erstellten Prototypen implementiert wurden. Aufgrund der zeitlichen Beschränkung konnte nur ein Teil der identifizierten Features auch mit dem Prototypen getestet werden.
Bei der Auswahl der Features, die implementiert wurden, wurde iterativ vorgegangen. Zunächst wurde ein grundsätzliches Feature-Set definiert, welches implementiert werden muss, um die Kernanforderung an die Software abbilden zu können. Dies war ein technischer Durchstich von den Rohdaten als ZIP-Archiv bis zur Suche innerhalb des visualisierten Netzwerks, dass die Rohdaten beschreiben. Nachdem dieser Durchstich implementiert war, wurde der Implementierungsaufwand im Vergleich zum Nutzen abgewogen und es wurden aufgrund dieser Bewertung zusätzliche Features ausgewählt, die daraufhin implementiert wurden.

Feature	Beschreibung	ist implementiert
F1	Daten-Upload	o
F2.1	Anzahl importierter Rohdaten	x
F2.2	Anzahl Knotenpunkte pro Entität	x
F2.3	Anzahl Kanten pro Kantentyp	x
F2.4	Anzahl an Interaktionen eines Users pro Typ	o
F2.5	Varianz der Interaktionshäufigkeit der User	o
F3	Zurücksetzen der Applikation	o
F4	Administrierung der Plattform	x
F5	Monitoring des Datenimports	x
F6	Suchmaschine	x
F7.1	Begrenzung des dargestellten Netzwerks	x
F7.2	Einflussnahme auf Netzwerk-Darstellung	x
F7.3	Mehr Information bei Hover über Knotenpunkt	x
F7.4	Unterscheidung Knotenpunkte über Farbe/ Größe	x
F7.5	Intuitive Navigation durch Netzwerk	o (teilweise impl.)
T1	Skalierbarer Datenimport	x (vorbereitet)
T2	Parallelbetrieb Datenimport und Datenanalyse	o
T3	Betrieb als Docker-Container	x

Tabelle 2: Zuordnung von Features->Prototyp

Wie zu sehen ist, wurden die Features F1, F2.4, F2.5 sowie T2 nicht implementiert und das Feature F7.5 nur teilweise. Der Grund dafür, F1 nicht zu implementieren, war eine Abwägung des Implementierungsaufwandes im Vergleich zur Relevanz in Bezug auf die übergreifende Fragestellung dieser Arbeit. Die Features 2.4 sowie 2.5 konnten nicht implementiert werden, da die zeitliche Rahmen für die Implementierung ausgeschöpft war. das Feature T2 wurde nicht implementiert, da die erarbeitete Architektur viel zu komplex für einen Prototypen ist. Da sich diese Anforderung aus der Architektur ergibt, war eine Implementierung unter der genannten Prämisse nicht notwendig.
Das Feature F7.5 wurde teilweise implementiert. Die Navigationsmöglichkeit Zoomen wurde komplett und die Neuwahl des dargestellten Ausschnittes des Graphen über die Suche, also teilweise, implementiert. Die Navigation mittels Verschiebens des Bildausschnittes konnte nicht implementiert werden, da der zeitliche Rahmen bereits ausgeschöpft war. Da bereits eine teilweise Navigation durch das Netzwerk mit den genannten Features möglich ist, wurde anderen Features der Vorrang gegeben, die sonst in keinster Weise im Prototypen implementiert worden wären.

7.2 Architektur des Prototypen

Im Vergleich zu der erarbeiteten Zielarchitektur wurde beim Prototyp ein anderer Ansatz gewählt. Da die erarbeitete Zielarchitektur den Anspruch hat, einer im professionellen Rahmen genutzten Software zu genügen sowie auf einer dedizierten Infrastruktur betrieben werden sollte, würde eine Implementierung dieser Architektur den Rahmen

dieser Arbeit überschreiten. Der Prototyp hat ferner die Primäraufgabe, die erarbeiteten Prozesse zum Datenimport, zur Datenverarbeitung sowie zur Visualisierung eines interaktiven Netzwerks zu validieren. Dafür bedarf es keiner perfekten Architektur, die tausende User gleichzeitig bedienen kann.
Der größte Unterschied ist der Verzicht auf eine Aufspaltung in mehrere Microservices. Dies reduziert die Komplexität enorm. Der Prototyp ist also von monolithischer Natur und vereint die Funktionalitäten der drei beschriebenen Microservices.
Bezüglich des SPA-Frameworks wurde sich für vue.js entschieden, da es sich um das schlankeste Framework der drei zur Auswahl stehenden handelt. Die Lernkurve während der Implementierung war dementsprechend geringer, auch, da viele Prozesse und Funktionen, die in den anderen Frameworks mit der Zeit dazu gekommen sind und somit nicht immer den gleichen Paradigmen des Rests des Frameworks folgen, direkt einheitlich implementiert wurden. Prominente Beispiele dafür sind Reactive Properties sowie State-Management.
Der Empfehlung, Spring für die Backend-Services zu verwenden, wurde gefolgt. Im Speziellen wurde Spring-Boot genutzt, um ein schnelleres Scaffolding des Prototypen zu erreichen.
Für die Visualisierung des Netzwerks wurde d3.js eingesetzt. Speziell sollte hiermit validiert werden, wie gut ein Zusammenspiel eines SPA-Frameworks mit einem komplexen Rendering-Framework wie d3.js funktioniert.

7.3 Detaillierte Implementierungsbeispiele

7.3.1 Job-Framework

Im Folgenden wird die Implementierung des Job-Frameworks , dass in Unterabschnitt 6.4 beschrieben wurde, vorgestellt.
Job

```
1  this.amountOfWorkDone = 0;
2  this.amountOfTotalWork = this.amountOfTotalWorkSupplier.get();
3  final long beginning = System.currentTimeMillis();
4
5  try {
6    this.execute();
7  } catch (final Exception exception) {
8    throw new RuntimeException(exception);
9  }
10
11 final long end = System.currentTimeMillis();
12 log.info("job completed! it took {}s", (end - beginning) / 1000);
```

Listing 1: Implementierung der Run-Methode eines Jobs

Feldname	**Typ**	**Beschreibung**
jobName	JobName	Job-Name als Enum, um Job eindeutig zuordenbar zu machen.
description	String	Eine kurze Beschreibung des Jobs.
mutexGroup	List<JobName>	Eine Auflistung von Jobs, die nicht parallel zu diesem Job ausgeführt werden dürfen.
amountOfTotalWorkSupplier	Supplier<Long>	Eine Supplier-Function, die, wenn ausgeführt, die Gesamtzahl an Arbeitspaketen dieses Jobs angibt.
jobStatus	JobStatus	Status des aktuellen Jobs als Enum.
completion	Double	Prozentualer Fertigstellungsgrad des Jobs zwischen 0-1
amountOfWorkDone	Double	Anzahl an bereits fertiggestellten Arbeitspaketen.

Tabelle 3: Felder und Beschreibung eines Jobs

In der abstrakten Klasse Job ist die Logik implementiert, die für jeden Job gelten soll. Ein Job implementiert hierbei das Runnable-Interface und kann somit asynchron in einem extra Thread ausgeführt werden. Um das Runnable-Interface zu erfüllen, muss eine run-Methode implementiert werden. Diese Implementierung ist in Listing 1 zu sehen. Die Felder und Erläuterungen sind in Tabelle 3 zu finden.
Jeder Job hat die Möglichkeit, seinen aktuellen Fortschritt zu tracken. Dafür wird intern eine Kennzahl, amountOfWorkDone, gemessen. Jedes mal, wenn der Job ein Arbeitspaket, dass je nach konkreter Job-Implementierung unterschiedlich definiert sein kann, abgeschlossen hat, wird dies durch einen Methodenaufruf auf der Job-Instanz in ebenjener Kennzahl festgehalten.
Weiterhin besitzt jeder Job eine abstrakte Methode: execute. Diese Methode wird von dem konkreten Job implementiert und beinhaltet die eigentliche Logik, die der Job ausführen soll.
Wird nun die run-Methode des Jobs ausgeführt werden zunächst die Kennzahl der bereits erfüllten Arbeitspakete zurückgesetzt und die Kennzahl, die die Gesamtzahl an Arbeitspaketen angibt, gefüllt. Daraufhin wird die Startzeit des Jobs in einer lokalen Variable gespeichert. Nun wird die execute-Methode ausgeführt. Dies geschieht in einem try-catch-Block, der sämtliche Exceptions abfängt und diese als Runtime-Exceptions rethrowed. Hierdurch wird sichergestellt, dass keine checked-Exceptions aus der Jobausführung ausbrechen und Einfluss auf die grundlegende Programmausführung nehmen können.
Abschließend wird der Endzeitpunkt des Jobs gemessen und die Laufzeit des Jobs gelogged.

JobService

Die JobService-Klasse ist ein spring-Service. Sie hat die Verantwortung über das Management der verschiedenen Jobs der Applikation. Sie beinhaltet eine Referenz auf eine Instanz aller konkreten Klassen, die die abstrakte Basisklasse Job erweitern. Weiterhin hält sie eine Referenz auf alle aktuell laufenden Jobs.
Der Service bietet über die Methode startJob die Möglichkeit, einen Job anhand seines Namens zu starten. Diese Methode wird vo der REST-API genutzt. Hierbei kann es zu 2 Exceptions kommen, die dann zu entsprechenden Http-Response-Codes führen, die wiederum der Konsument der REST-API, z.B. das Frontend, nutzen kann, um dem User eine entsprechende Fehlermeldung anzuzeigen. Eine Exception, JobRunningException, tritt auf, wenn der entsprechende Job bereits läuft. Per Definition kann ein Job nicht mehrfach gleichzeitig laufen, da dies zu verschiedenen Problemen, die aus Nebenläufigkeit, dem Gleichzeitigen Bearbeiten der gleichen Entität oder durch Race-Conditions resultieren können, führen kann. Die andere Exception, JobInRunningMutexGroupException, tritt auf, wenn bereits ein Job läuft, der angegeben hat, dass der Job, der ausgeführt werden soll, nicht parallel zu ebenjenem Job ausgeführt werden darf, oder wenn der Job, der ausgeführt werden soll, angegeben hat, dass ein bereits laufender Job nicht parallel zu ebenjenem Job ausgeführt werden darf.
Wird keine der beiden Exceptions geworfen wird eine Referenz auf den entsprechenden Job in der Liste der gerade laufenden Jobs hinterlegt, der Status des Jobs auf running gesetzt, der Job über die run-Methode in einem neuen Thread gestartet sowie der User über das erfolgreiche Starten des Jobs via entsprechendem Http-Response-Code informiert.

7.3.2 Rohdatenimport

Bei dem Rohdatenimport handelt es sich um den Prozessschritt, der die vorliegenden Daten als Zip-Archiv empfängt und diese in die vorhandene Mongo-Datenbank importiert. um den Import in die Mongo-Datenbank vorzunehmen wird die Bibliothek spring-boot-starter-data-mongodb genutzt. Diese bietet eine Verknüpfung zwischen dem Datenmodell und dem mongoDB-Treiber für Java und bietet unter anderem object-mapping, query-derivation, auditing sowie transaction-handling als Features und ist somit von großem Mehrwert.
Der eigentliche Job wurde als Job innerhalb des Job-Frameworks implementiert. Weiterhin wurde der RawDataImportService implementiert, der die Verarbeitung des Zip-Archives übernimmt. Innerhalb der Application-Properties lässt sich der Pfad zum Zip-Archiv konfigurieren. Momentan wird das Archiv mit der Applikation zusammen ausgeliefert.
Die Klasse RawDataImportService hat 2 Funktionen, die von der Applikation genutzt werden können:

- Das Abrufen der eingelesenden Dateien des Zip-Archives
- Das Abrufen der Anzahl an Dateien, die das Zip-Archiv enthält

```
Stream<Map<String, Object>>
```

Listing 2: Rückgabewert der Methode getRawData des RawDataImportService

Der Rückgabewert wurde bewusst so gewählt und repräsentiert einen kontinuierlichen Daten-Strom, dessen Inhalte jeweils eine Map sind, die eine geladene Entität darstellt und deren Keys die Feldnamen als String und deren Values die eigentlichen Werte des jeweiligen Feldes als Object sind. Nachgelagert können diese dann in die konkreten Java-Typen konvertiert werden. Da die Streaming-API von Java verwendet wird kann sogar mit geringem Aufwand eine kontinuierliche Datenverarbeitung der Daten der Twitter-API via http keep-alive Connections implementiert werden, sodass neue Daten vollautomatisch konsumiert und importiert werden [Twia].

7.3.3 Import in Graphdatenbank

Für den Import der Rohdaten in die Graphdatenbank wurde die Bibliothek spring-data-neo4j genutzt. Die Wahl fiel auf die genannte Bibliothek, da so Synergieeffekte zwischen spring-data-mongodb und spring-data-neo4j genutzt werden können. Weiterhin bietet spring-data im generellen eine exellente Integration in spring.
Spring-data-neo4j bietet den entscheidenden Vorteil, dass Relationen zwischen den verschiedenen Knotenpunkten direkt im Java-Modell angegeben werden können. Somit kann ein objektorientierter Ansatz in der Applikation beibehalten werden und die Transformation dieser nach objektorientierung modellierten Entitäten in für die Graphdatenbank optimierten Knoten und Kanten wird durch die Bibliothek gekapselt.
Für jeden Entitätstyp in der Graphdatenbank wurde ein Job mithilfe des Job-Frameworks implenentiert. Die Implementierung der Jobs wurde nach Unterunterabschnitt 6.7.3 durchgeführt.

ImportGraphUsersJob

Der ImportGraphUsersJob ist dafür verantwortlich, die User-Rohdaten in User-Knotenpunkte umzuwandeln und diese inklusive ihrer Relationen in die Graphdatenbank zu importierten. Dabei wird für jeden User der Rohdaten der Code-Block Listing 3 ausgeführt.

```
if (userService.findByRawId(user.getRawId()).isEmpty()) {
    final List<Tweet> allTweetsByUser = tweetService
            .findAllTweetsByUserId(user.getRawId());
    allTweetsByUser.forEach(user::addTweet);

    rawDataService.getAllRetweetsByUserId(user.getRawId())
            .forEach(reTweet ->
                    tweetService.findByRawId(reTweet.getReference())
                            .ifPresent(user::addReTweet));
```

```
10
11      userService.save(user);
12
13      tweetService.findAllByMentionedIdsContaining(user.getRawId())
14              .forEach(tweet -> {
15                  tweet.addMentionedUser(user);
16                  tweetService.save(tweet);
17              });
18  }
```

Listing 3: Algorithmus, der für den Import der User-Rohdaten genutzt wird

ImportGraphTweetsJob

Der ImportGraphTweetsJob ist dafür verantwortlich, die Status-Rohdaten in Tweet-Knotenpunkte umzuwandeln und diese inklusive ihrer Relationen in die Graphdatenbank zu importierten. Dabei wird für jeden Status der Rohdaten der Code-Block Listing 4 ausgeführt.

```
1  if (tweetService.findByRawId(tweet.getRawId()).isEmpty()) {
2      tweet.getMentionedIds()
3              .forEach(userId -> userService.findByRawId(userId)
4                      .ifPresent(tweet::addMentionedUser));
5
6      tweetService.save(tweet);
7
8      userService.findByRawId(tweet.getUserId())
9              .ifPresent(authorOfThisTweet -> {
10                 authorOfThisTweet.addTweet(tweet);
11                 userService.save(authorOfThisTweet);
12             });
13
14     tweetService.findByQuoteTargetId(tweet.getRawId())
15             .ifPresent((quoter) -> {
16                 quoter.setQuoteTarget(tweet);
17                 tweetService.save(quoter);
18             });
19
20     tweetService.findByReplyTargetId(tweet.getRawId())
21             .ifPresent(reply -> {
22                 reply.setTarget(tweet);
23                 tweetService.save(reply);
24             });
25
26     rawDataService.getAllRetweetsByReference(tweet.getRawId())
27             .forEach(reTweet ->
28                     userService.findByRawId(reTweet.getUserId())
29                             .ifPresent(user -> {
30                                 user.addReTweet(tweet);
```

```
31                                      userService.save(user);
32                                  }));
33  }
```

Listing 4: Algorithmus, der für den Import der Status-Rohdaten genutzt wird

ImportGraphRetweetsJob

Der ImportGraphRetweetsJob ist dafür verantwortlich, die Retweet-Rohdaten in retweets-Relationen umzuwandeln und diese in den Graphen einzufügen. Dabei wird für jeden Retweet der Rohdaten der Code-Block Listing 5 ausgeführt.

```
1 userService.findByRawId(reTweet.getUserId())
2         .ifPresent(user ->
3                 tweetService.findByRawId(reTweet.getReference())
4                         .ifPresent(tweet -> {
5                             if (!user.getReTweets().contains(tweet)) {
6                                 user.addReTweet(tweet);
7                                 userService.save(user);
8                             }
9                         }));
```

Listing 5: Algorithmus, der für den Import der Retweet-Rohdaten genutzt wird

ImportGraphRepliesJob

Der ImportGraphRepliesJob ist dafür verantwortlich, die Reply-Rohdaten in replies_to-Relationen umzuwandeln und diese in den Graphen einzufügen. Dabei wird für jede Reply der Rohdaten der Code-Block Listing 5 ausgeführt.

```
1  if (tweetService.findByRawId(tweet.getRawId()).isEmpty()) {
2      tweet.getMentionedIds()
3              .forEach(userId -> userService.findByRawId(userId)
4                      .ifPresent(tweet::addMentionedUser));
5      tweetService.findByRawId(tweet.getReplyTargetId())
6              .ifPresent(tweet::setTarget);
7
8      tweetService.save(tweet);
9
10     userService.findByRawId(tweet.getUserId())
11             .ifPresent(authorOfThisTweet -> {
12                 authorOfThisTweet.addTweet(tweet);
13                 userService.save(authorOfThisTweet);
14             });
15
16     rawDataService.getAllRetweetsByReference(tweet.getRawId())
17             .forEach(reTweet ->
18                     userService.findByRawId(reTweet.getUserId())
19                             .ifPresent(user -> {
20                                 user.addReTweet(tweet);
21                                 userService.save(user);
```

```
22                                  }));
23 }
```

Listing 6: Algorithmus, der für den Import der Reply-Rohdaten genutzt wird

ImportGraphQuotesJob

Der ImportGraphQuotesJob ist dafür verantwortlich, die Quote-Rohdaten in quotes-Relationen umzuwandeln und diese in den Graphen einzufügen. Dabei wird für jede Quote der Rohdaten der Code-Block Listing 7 ausgeführt.

```
1  if (tweetService.findByRawId(tweet.getRawId()).isEmpty()) {
2      tweet.getMentionedIds()
3              .forEach(userId -> userService.findByRawId(userId)
4                      .ifPresent(tweet::addMentionedUser));
5
6      tweetService.save(tweet);
7
8      userService.findByRawId(tweet.getUserId())
9              .ifPresent(authorOfThisTweet -> {
10                 authorOfThisTweet.addTweet(tweet);
11                 userService.save(authorOfThisTweet);
12             });
13
14     tweetService.findByRawId(tweet.getQuoteTargetId())
15             .ifPresent(tweet::setQuoteTarget);
16
17     rawDataService.getAllRetweetsByReference(tweet.getRawId())
18             .forEach(reTweet -> userService.findByRawId(reTweet.getUserId
                   ↪ ())
19                     .ifPresent(user -> {
20                         user.addReTweet(tweet);
21                         userService.save(user);
22                     }));
23 }
```

Listing 7: Algorithmus, der für den Import der Quote-Rohdaten genutzt wird

7.4 Laufzeiten des Datenimports

In Tabelle 4 sind die Laufzeiten der Jobs zu finden. Dabei ist zu beachten, dass diese Messwerte mit einem Computer mit den folgenden Hardware-Spezifikationen erzeugt wurden, der während der Laufzeit wenig bis moderat anderwaltig genutzt wurde.

- Betriebssystem: Windows 10 x64
- CPU: Intel Core i9-9900KF @3.6 GHz
- Arbeitsspeicher: 64 GB @ 2666 MHz

- Grafikkarte: NVIDIA GeForce RTX 2070 SUPER
- Datenträger: Samsung SSD 970 EVO Plus 500GB

Die neo4j-Datenbank wurde dabei so konfiguriert, dass bis zu 8 GB Arbeitsspeicher für den Java Heap verwendet werden dürfen und dass bis zu 24GB Arbeitsspeicher für den PageCache genutzt werden dürfen.
Weiterhin muss beachtet werden, dass es eine starke Abhängigkeit zwischen Laufzeit und Ausführungsreihenfolge gibt. Da, wie bereits beschrieben, eine Relation in der Graphdatenbank nur gebildet wird, wenn beide Entitäten vorhanden sind, ist die Ausführung des ersten Jobs für die Graphdatenbank deutlich schneller, da mit diesem noch keine Relationen gebildet werden können. Die Jobs wurden in der Reihenfolge ausgeführt, in der sie in Tabelle 4 aufgelistet sind.

Jobname	**Laufzeit**	**Arbeitsvolumen**
RawDataImportJob	3,23 Minuten	4651068 Rohdatensätze
UserImportJob	76,38 Minuten	914257 User-Rohdatensätze
TweetImportJob	24,8 Stunden	1099803 Status-Rohdatensätze
RetweetImportJob	125,28 Minuten	1415609 Retweet-Rohdatensätze
ReplyImportJob	21,8 Stunden	876276 Reply-Rohdatensätze
QuoteImportJob	3,84 Stunden	345123 Quote-Rohdatensätze

Tabelle 4: Laufzeiten der verschiedenen Jobs

Wie zu sehen ist, sind die Laufzeiten teilweise sehr lang. Um dies genauer zu untersuchen, wurden weitere Laufzeit-Messungen mit unterschiedlichem Volumen durchgeführt. In Abbildung 15 Ist die summierte Laufzeit aller Jobs für die verschiedenen Volumen zu finden. In Abbildung 16 ist das Rohdatenimportvolumen pro Sekunde zu sehen. In Abbildung 17 ist das Graphentitätenimportvolumen pro Sekunde abgebildet.

Die Messergebnisse lassen auf einen exponentiellen Anstieg der Laufzeit schließen. Dies wird durch eine Trendlinie in Abbildung 16 ersichtlich, die ein Bestimmtheitsmaß von über 98% aufweist. Gleichzeitig wird in den anderen beiden Abbildungen ersichtlich, wie das importierte Volumen pro Sekunde linear abnimmt. Es fällt weiterhin auf, dass das Volumen an importierten Rohdaten pro Sekunde schneller sinkt, als das Volumen an importierten Graphentitäten.
Leider bietet der neo4j-Java-Treiber noch keine Möglichkeit, ein Batch-Insert durchzuführen. Somit müssen für alle Entitäten, die in den Graph eingefügt werden sollen, eine Reihe von Requests gegen die Datenbank ausgeführt werden: Ein Request, um die Entität einzufügen sowie jeweils ein weiterer Request für jede Beziehung, die zwischen dem Datensatz und bereits bestehenden Datensätzen hergestellt werden soll. Im Beispiel von spring-data-neo4j ist das Problem sogar noch gravierender: Da ein Object-Graph-Mapper (OGM) eingesetzt wird, müssen die Knotenpunkte, zu denen eine Beziehung gebildet werden soll, erst aus der Datenbank geladen werden, um dann anschließend

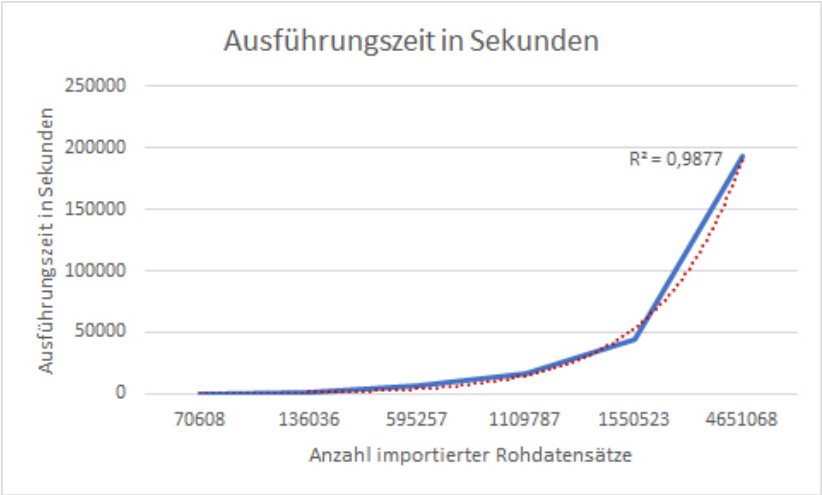

Abbildung 15: Messung der Dauer des Importsfür unterschiedliche Volumen an Rohdaten

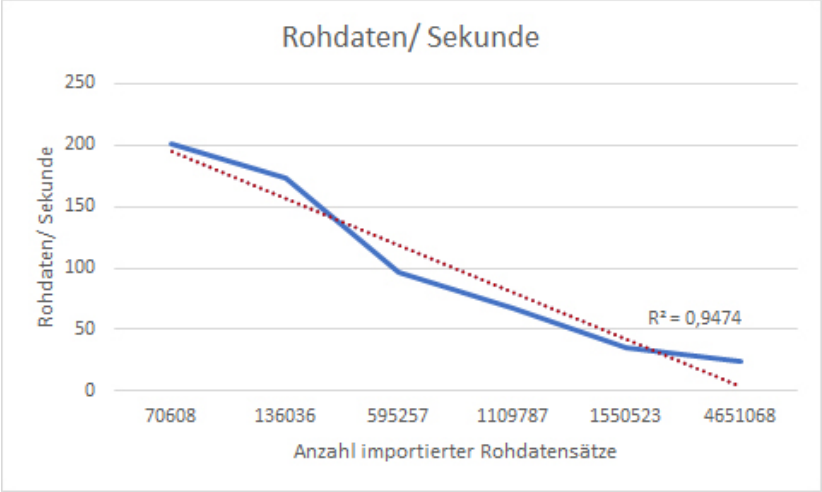

Abbildung 16: Messung der Importgeschwindigkeit in Rohdaten pro Sekunde für unterschliedliche Volumen an Rohdaten

zunächst eine Beziehung zwischen den Objekten in Java zu bilden, bevor diese dann wieder dem OGM übergeben werden können, der daraufhin die Insert-Queries bildet. Daher war es nicht möglich, eine Laufzeitverbesserung mit diesem Ansatz zu erzielen. Er eignet sich also besser, um einen kontinuierlichen Strom an Daten zu importieren und nicht einen bereits über eine lange Zeit gesammelten Datensatz. Hierfür ist ein Batch-Upload besser geeignet. Neo4j bietet beispielsweise die Möglichkeit, ein CSV-Dokument zu importieren. Ein Test mit einem CSV-Dokument, dass ein vergleichbares Volumen wie der vorliegende Datensatz hatte, ergab eine Laufzeit von 61,6 Minuten, was eine deutliche Verbesserung darstellt. Dieser Ansatz hat allerdings auch Nachteile, da hierbei beispielsweise Indexe erst nachträglich gebildet werden können und somit Anforderungen an Einzigartigkeit bestimmter Felder nicht mehr vor dem Einfügen geprüft werden können. Weiterhin müssten die Daten auch erst in das CSV-Format transformiert werden, was auch Aufwand erzeugen würde. Weiterhin akzeptiert diese

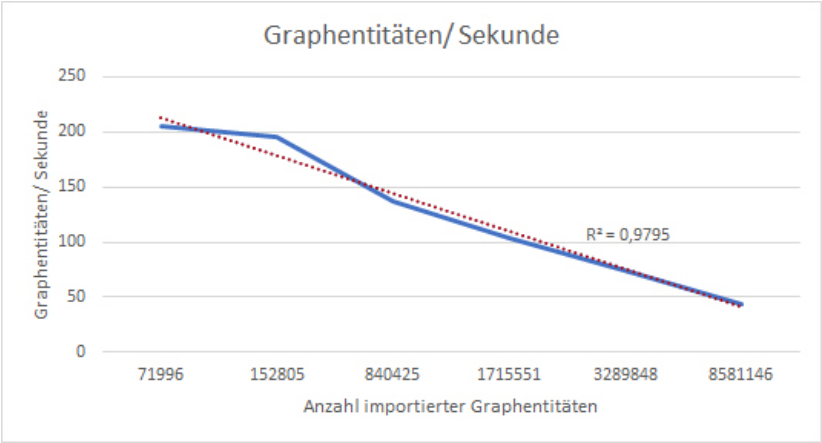

Abbildung 17: Messung der Importgeschwindigkeit in Graphentitäten pro Sekunde für unterschiedliche Volumen an Rohdaten

Schnittstelle die CSV-Datei nicht direkt, sondern benötigt eine URL, die auf die Datei verweist. Dies macht ein programmatisches Aufrufen der Schnittstelle mühsam. Es gibt in einer CSV-Datei auch keine Möglichkeit, den Typ eines Feldes anzugeben. Daher müssen alle Felder nochmal explizit in der Cypher-Query, die den Import anstößt, in den entsprechenden Typ konvertiert werden.

7.5 Bewertung der Implementierung

Bei der Kombination von vue.js sowie d3.js kam es zu einer Reihe von Problemen. Da vue.js ein komplexes Rendering-Framework ist, dass unter anderem ein komplettes Reactivity-System implementiert, mithilfe dessen Komponenten im Frontend automatisch neu gerendert werden, wenn sich die Datengrundlage verändert, ist vue.js nicht dafür ausgelegt, massenhaft Komponenten zu rendern. Dies ist jedoch notwendig, denn das darzustellende Netzwerk besteht aus einer Vielzahl von Knotenpunkten und Kanten. Eine Performance-Auswertung, die mit Google Chrome angefertigt wurde, ist in Abbildung 18 zu finden. Dabei wurde versucht, ein Netzwerk mit 10000 Knotenpunkten zu rendern. Da nicht alle identifizierten Features implementiert werden konnten, besteht insbesondere im Bereich der Navigation durch das Netzwerk ein Mangel. Es ist zwar möglich, dass ein User mithilfe der Suchfunktion einen geeigneten Einstiegspunkt ins Netzwerk findet, allerdings gibt es für ihn keinen einfachen Weg, einen Knotenpunkt im Netzwerk auszuwählen und diesen zum neuen Einstiegspunkt zu machen, ohne seine Daten in die Suche einzugeben. Ein weiterer Punkt, der die Navigation einschränkt, ist, dass sich der Bildausschnitt nicht verschieben lässt.
Von diesen Punkten abgesehen funktioniert der Prototyp wie erwartet. Inofern validiert die Implementierung den Implementationsleitfaden in Abschnitt 6 und damit in gewisser Weise aus die erarbeitete Architektur, auch wenn der Prototyp teilweise aus Gründen der Komplexitätsverringerung von Dieser abweicht. Gerade das Konzept mit den beiden Datenbanken hat sich während der Implementierung als wertvoll heraus-

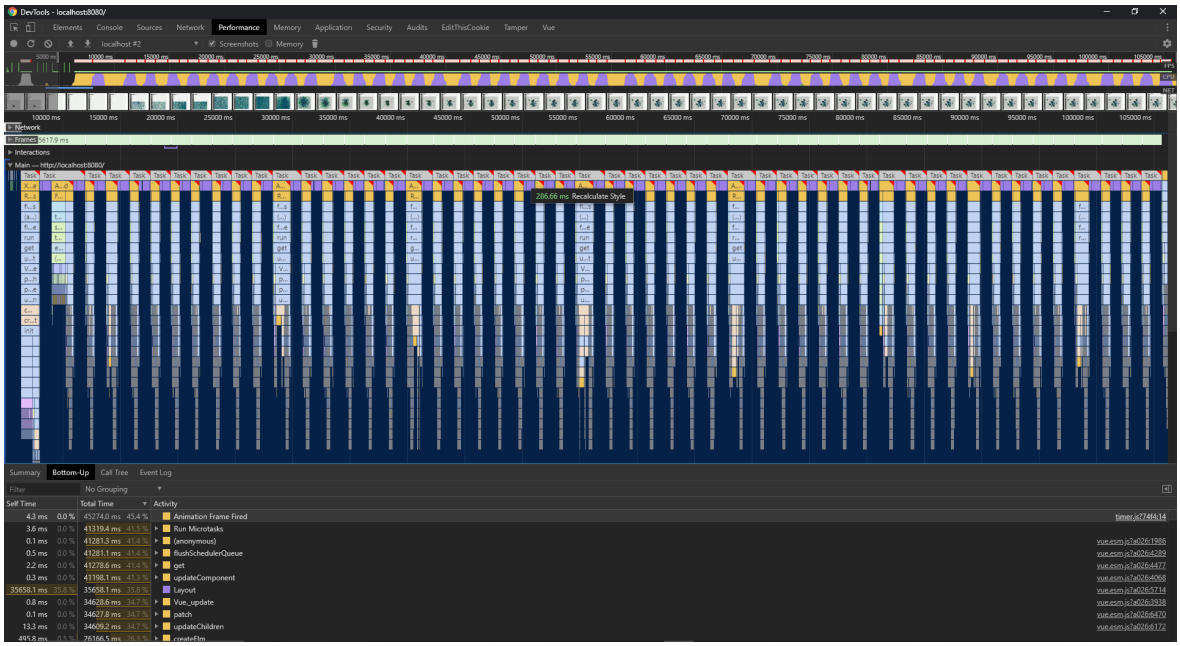

Abbildung 18: Screenshot der Chrome-Devtools während des Renderns von 10000 Knotenpunkten

gestellt, denn so konnte der Datenimportprozess in mehrere Teilschritte aufgespalten werden, die nicht jedes Mal wiederholt werden mussten, wenn eine neue oder erweiterte Implementierung eines Teilschrittes getestet wurde.

8 Validierung der Ergebnisse

Um die Ergebnisse zu validieren wurde ein UX-Test mit einigen Fachexperten durchgeführt. Diesen wurd die erstellte Software grundlegend erklärt und sie wurden anschließend gebeten, diese auszuprobieren und zu bewerten.

8.1 Methodik

Der Test wurde mit jedem Probanden einzeln durchgeführt. Dabei wurde der Proband in einen Raum gerufen, in dem ein Laptop mit einer installierten und laufenden Instanz des in Abschnitt 7 beschriebenden Prototyps stand. Die Datengrundlage war ein Sample des Gesamtdatensatzes mit einem Umfang von 10%. Dieses Sampling war notwendig, da der Proband auch die Datenimportprozesse ausprobieren sollte, und dies mit einem ganzen Datensatz aufgrund der Dauer des Imports nicht möglich gewesen wäre.
Nun wurde dem Probanden zunächst kurz der generelle Problemraum erklärt. Dabei wurde auf Nachfragen eingegangen. Der Proband sollte zuversichtlich sein, einen Überblick über den Problemraum zu haben. Anschließend wurde dem Probanden die vorliegende Software erläutert. Dabei wurde grob erklärt, welche Funktionen es gibt, wo diese zu finden sind, und wie die generellen Arbeitsprozesse mit dem Prototypen funktionieren.
Anschließend wurde dem Probanden ein Fragebogen nach dem System Usability Scale-Prinzips (SUS) [Jor+96] vorgelegt. Dieser ist in Unterabschnitt 10.1 zu finden. Wie zu sehen ist, beinhaltet der Fragebogen ein einleitendes Szenario. Hier wird dem Probanden anhand eines Beispiels nochmal Zusammenfassend der Problemraum aufgezeigt. Das Beispiel in dem Szenario dient dazu, dass sich der Proband besser in ein mögliches Nutzungsszenario des Prototypen heineindenken kann. Weiterhin folgt nach der eigentlichen Befragung noch ein Raum für Kommentare, die ein Proband noch mitteilen möchte.
SUS-Fragebögen sollen eine Skala nach dem Likert-Prinzip enthalten. Häufig werden hierbei Likert-Skalen mit fünf oder sieben Möglichkeiten angegeben, wobei die mittlere Option dann jeweils weder Zustimmung noch Ablehnung angibt. Dieses Modell wurde auch für diesen Fragebogen gewählt. Eine mittlere Option kann zwar zu einer Neigung der Probanden zur mittleren Option führen, allerdings führt ein Weglassen zu anderen Problemen, wie geringerer Verlässlichkeit, da Probanden, die durch diese Option korrekt repräsentiert wären, zufällig eine andere Option, die nahe an der Mitte liegt, wählen [MW10].
Die Befragung wurde mit 8 unterschiedlichen Personen durchgeführt. Davon arbeiten alle beruflich im Data-Science-Umfeld und bringen somit eine Menge an Hintergrundwissen bzgl. Analyse von Massendaten mit. Aufgrund der geringen Teilnehmerzahl

sollten vor Allem die Ergebnisse gewertet werden, die eine besonders geringe Varianz in den Antworten aufweisen, da eine hohe Varianz auf eine ungenügende statistische Relevanz schließen lässt.
Für die Auswertung wurden jeder Antwort numerische Werte von 1-5 zugewiesen. Abhängig davon, ob eine Frage positiv oder negativ formuliert war, beginnt die jeweilge Skala der Frage entweder bei 1 oder bei 5, wobei die 1 immer die negativste und 5 immer die positivste Antwort kennzeichnet.

8.2 Ergebnisse

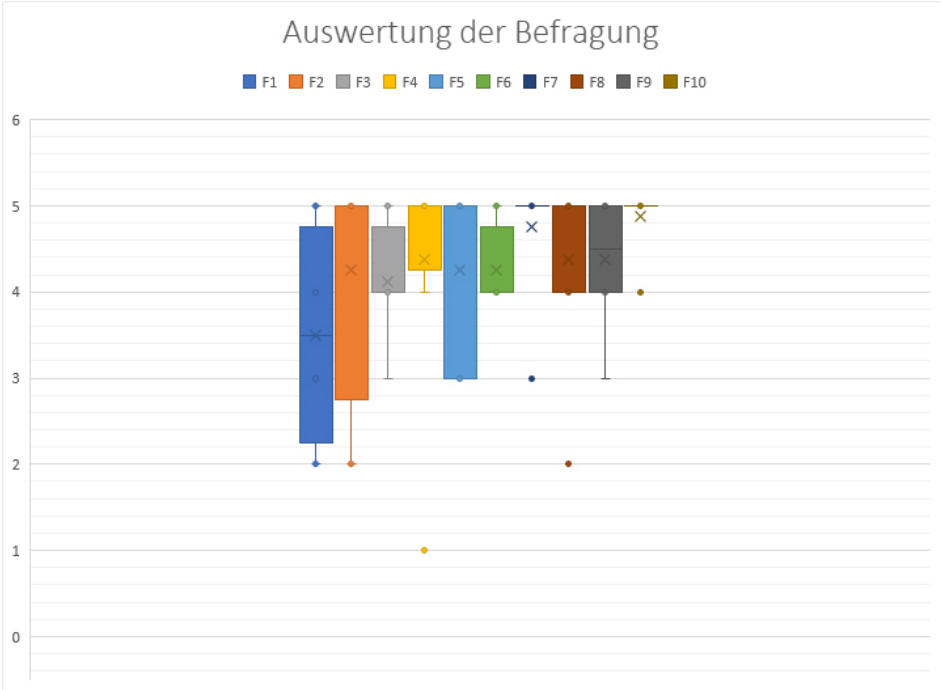

Abbildung 19: Auswertung der Befragung - Ergebnisse

Die Rohdaten der Ergebnisse sind in Unterabschnitt 10.2 zu finden. In Abbildung 19 findet sich eine Zusammenfassung der Ergebnisse. Die Varianz innerhalb der jeweiligen Antwort ist in Abbildung 20 zu finden. Die Metriken F1-F10 stellen die jeweilige Frage im Fragebogen dar. Wie zu sehen ist, besitzen insbesondere die Antworten F3, F6 und F10 eine geringe Varianz innerhalb der Antworten. Insofern scheinen sich die Teilnehmer sehr einig gewesen zu sein.
In Frage 3 wurde gefragt, wie einfach die Benutzung des Systems ist. Die Befragten waren sich sehr einig in der Antwort, dass das System sehr einfach zu benutzen ist. Dies wird zusätzlich dadurch gestützt, dass die Frage 10, die fragt, ob ein Befragter viel Hintergrundwissen benötigte, bevor er mit dem Prototypen arbeiten konnte, ebenfalls sehr einig damit bewertet wurde, dass die Probanden kaum zusätzliches Wissen benötigten, um das System benutzen zu können. Die Frage F6 fragt, ob es viele Inkonsistenzen im System gab. Diese Aussage wurde sehr geeint stark abgelehnt.

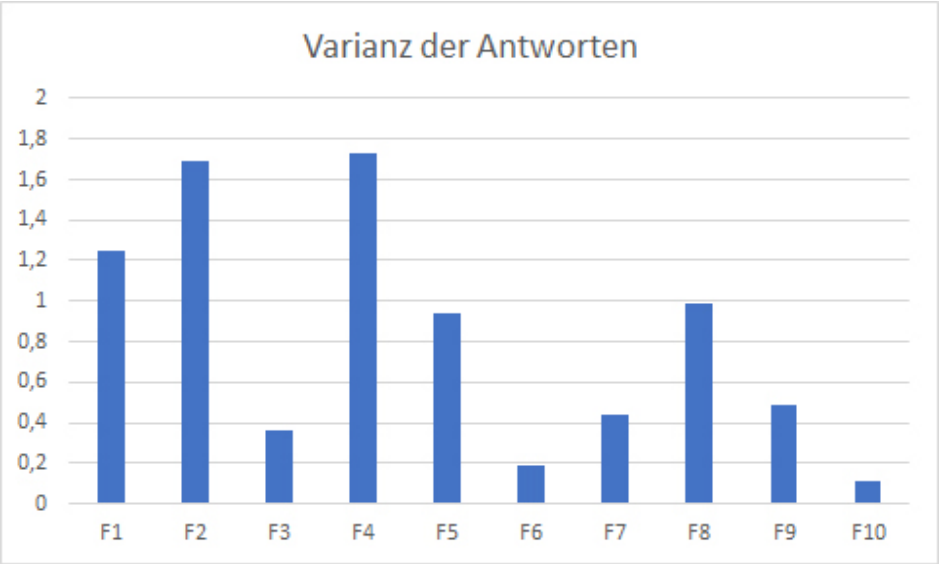

Abbildung 20: Auswertung der Befragung - Varianz

Die Antworten mit der größten Varianz sind F1, F2 und F4. Die Teilnehmer scheinen sich sehr uneinig gewesen zu sein. Dies kann darauf hindeuten, dass die Antworten sehr subjektiv waren, dass die Frage sehr polarisierend war oder einfach, dass die Teilnehmerzahl zu gering war.
Bei Frage F1 wurde gefragt, ob das System einen regelmäßigen Nutzen für die Arbeit, die im einleitenden Szenario beschrieben wurde, liefert. Eine mögliche Erklärung für die hohe Varianz ist die Tatsache, dass im Prototypen nicht alle Features implementiert wurden. Dies kann dazu führen, dass nicht alle Arbeitsabläufe, die mithilfe des Prototypen unterstützt werden sollen, für den Probanden bewertbar sind. Eine weitere Möglichkeit ist, dass das einleitende Szenario die Aufgaben, die mit dem Prototypen gelöst werden sollen, nur grob beschreibt. Insofern obliegt es zu einem gewissen Maße der Interpretation der Probanden, ob und wie diese Arbeit nun mithilfe des Prototypens unterstützt werden kann.
Die Frage F2 fragt, ob das System unnötig komplex ist. Die hohe Varianz kann hier damit erklärt werden, dass eine Komplexitätsbewertung in der zur Verfügung stehenden Zeit nicht ausreichend möglich war. Zur Debatte steht auch, was die Probanden genau unter unnötig verstanden haben. Vermutlich hat jeder Proband ein anderes Maß an Komplexitätstoleranz, was wiederum die hohe Varianz erklären würde. Zu beachten ist allerdings, dass die Antworten für F2 sechs mal starke Ablehnung und nur zwei mal starke Zustimmung betragen haben.
F4 fragt schließlich, ob für die Benutzung der Software technischer Support notwendig ist. Hierbei antwortete ein Teilnehmer mit starker Zustimmung, ein Teilnehmer mit schwachem Widerspruch und 6 Teilnehmer mit starkem Widerspruch. Es handelt sich somit um einen Ausreißer.
In den abgegebenen Kommentaren war das meiste Lob für die physikalische Simulation

und das dazu gehörige Look&Feel sowie für die Tatsache, dass der Prototyp Data Scientists in ihrer Rolle ermächtigt, selbstständig Einfluss auf das Layout sowie Einblick in die Datenimportsprozesse zu nehmen. Die Kritik hat sich vor allem auf zwei Punkte konzentriert: Die Home-Seite würde nicht benötigt werden und der Funktionsumfang sei nicht groß genug. Dabei wurde sich im Speziellen die Möglichkeit gewünscht, eine direkte Eingabemöglichkeiten für Queries gegen die Graphdatenbank zu haben.

8.3 Einordnung der Ergebnisse

Aufgrund der bereits diskutierten geringen Teilnehmerzahl ist das Ergebnis nicht so aussagekräftig, wie es sein könnte. Obwohl alle Probanden Experten in der Domäne Data Science sind, wären mehr Teilnehmer für wirklich aussagekräftige Ergebnisse notwendig gewesen.
Durch die Kommentare wurden einige Umfrageergebnisse nochmal deutlicher und verständlicher. Einige der Ergebnisse lassen sich nutzen, um Teile des Prototypen zu bewerten.

8.4 Zusammenfassung der Ergebnisse

Da alle drei Fragen, die die geringste Varianz hatten, jeweils die intuitive und einfache Bedienung hervorheben, scheint der Prototyp hiermit zu validieren, dass das entwickelte Konzept zur Visualisierung einer interaktiven Repräsentation eines Großgraphen taugt. Hierbei ist zu beachten, dass der Prototyp noch nicht alle fachlichen Anforderungen erfüllt, was auch von einigen Probanden bemerkt wurde. Dies könnte ebenfalls Einfluss auf die Intuitivität haben.
Kontrovers wurde der tatsächliche Nutzen des Prototypen in Bezug auf das Eingangsszenario bewertet. Dies kann ein weiteres Zeichen dafür sein, dass sich die Probanden noch mehr implementierte Features gewünscht hätten.

9 Zusammenfassung

Zusammenfassend lässt sich feststellen, dass im Bereich der Visualisierung von Massendaten noch viele Fragestellungen ohne zufriedenstellende Antwort existieren. Analysen der Daten von Public APIs sozialer Netzwerke können zu wichtigen Erkenntnissen im Bereich der Sozialforschung führen. Gerade im Bereich der Erkennung von Verhaltensmustern ist eine Netzwerkanalyse ein sehr mächtiges Tool.
Twitter ermöglicht es seinen Nutzern durch eine Vielzahl von Interaktionsmöglichkeiten in Kontakt zu einander zu treten. Dies findet sich auch in den Daten wieder, da diese eine stark vernetzte Struktur aufweisen.
Die Darstellung von Massendaten kann nicht daran vorbei führen, die vorliegenden Daten vor der Anzeige entweder zu filtern oder zu aggregieren. Ein Versäumnis dessen führt zwangsläufig dazu, dass die Übersichtlichkeit so stark leidet, dass eine Analyse unmöglich wird. Für ein komplettes Analysesystem werden viele Features neben den eigentlichen Tools für die Datenanalyse benötigt. Gerade der Datenimport und die Datenverarbeitung, bevor die Daten überhaupt analysierbar sind, nimmt einen großen Teil ein. Nur ein bereinigter, normalisierter Datensatz, der alle notwendigen Informationen enthält, eignet sich für die Analyse.
Stark vernetzte Massendaten lassen sich besonders gut durch eine Netzwerkdarstellung visualisieren. Dabei ist eine primäre Fragestellung, wie das Layouting funktionieren soll. Dabei werden häufig Simulationen anhand eines physikalischen Kräftemodells angewendet. Dabei sollte das initiale Layout nicht final sein. Vielmehr sollten Wege geschaffen werden, durch die ein User direkten Einfluss auf das Layout ausüben kann, beispielsweise, in dem er einzelne Knotenpunkte so anordnet, wie er es präferiert.
Bei der Darstellung sollte viel Wert auf Übersichtlichkeit gelegt werden. Text-Labels beispielsweise dauerhaft in den Graphen zu rendern führt zu Frustration bei dem Anwender, da ihn die meisten dieser Labels nicht zeitgleich interessieren. Deutlich geschickter ist es, es durch verschiedene Interaktionsmechanismen zu ermöglichen, dass ein User nur die Informationen angezeigt bekommt, die ihn auch gerade interessieren. Dadurch wird die Übersichtlichkeit im Netzwerk maximiert, dass gerade im Kontext der Darstellung von Massendaten essentiell ist.
Damit sich ein User im gesamten Netzwerk zurecht findet, ist es wichtig, dass er sich durch verschiedene Interaktionsmöglichkeiten durch das Netzwerk bewegen kann. Dabei sollte es sowohl möglich sein, den aktuell angezeigten Ausschnitt des Netzwerks neu zu setzen, als auch, den aktuellen Ausschnitt durch bekannte Interaktionsmechanismen wie Zooming oder Panning zu modifizieren.
Eine der zentralen Kernfragen, die mit dieser Arbeit beantwortet werden sollte, war, wie eine optimale Architektur für solch eine Analysesoftware aussieht. Hier wurde eine komplexe Architektur aus mehreren Microservices konzipiert, die dank der Mongo-DB sowie der neo4j-Graphdatenbank eine solide Grundlage für den Datenimport sowie

Analysen jeglicher Art besitzt. Durch den Einsatz von State-of-the-art-Technologien wie Spring-Boot oder eines SPA-Frameworks verspricht die Architektur Zukunftssicherheit und Robustheit.
Durch die Konzeption des Frontends als mehrere Micro-Frontends wird die Entkopplung nach fachlicher Domäne, die im Backend vorgenommen wurde, auch für das Frontend ermöglicht. Es wurde weiterhin ein Deployment-Prozess für ebenjenes Frontend entwickelt, der durch Cache-Busting und atomare Deployments von Backend und Frontend die notwendigen Voraussetzungen erfüllt, die unabhängige Deployments und ein unabhängiger Betrieb an den Prozess stellen.
Es wurde weiterhin ein simples Datenmodell erarbeitet, dass die komplette Komplexität der vorliegenden Rohdaten abbilden kann. Weiterhin ist das Datenmodell in jeglicher Weise erweiterbar.
Für die verschiedenen identifizierten Features, die für eine Analyse der Twitter-Daten notwendig sind, wurde jeweils ein detailliertes Implementierungskonzept entwickelt, dass die erarbeitete Architektur respektiert. Insbesondere hervorzuheben ist hierbei das konzipierte Job-Framework, dass die Grundlage für sämtliche asynchrone Datenimport- und Analyseprozesse im Backend ist. Durch die generische Natur des Frameworks ist dieses zudem hochgradig erweiterbar und setzt somit einen wichtigen Grundstein für die Weiterentwicklung der Anwendung. Weiterhin wurde ein performanter und skalierbarer Prozess für die Darstellung der Daten als Netzwerk erarbeitet. Für den Rohdatenimport sowie den Graphdatenimport wurden zudem Prozesse definiert, die diese Tätigkeiten ausführlich beschreiben.
Mit dem Prototypen wurden eine Vielzahl von Features implementiert und somit validiert. Aufgrund der prototypischen Natur der Implementierung konnte nicht die komplette Architektur abgebildet werden. Da jedoch die Validierung der identifizierten Features im Vordergrund stand, hat der Prototyp damit seine Primäraufgabe erfüllt.
Durch den durchgeführten UX-Test mit einigen Fachexperten konnte zusätzliche Sicherheit bezüglich der Validierung gewonnen werden. Die Probanden haben jeweils einen SUS-Fragebogen ausgefüllt und haben dabei vor allem die intuitive Bedienbarkeit des Prototypen hervorgehoben. Weiterhin haben sie teilweise weitere Features vermisst, was darauf schließen lässt, dass das implementierte Konzept noch nicht den vollen erwarteten Funktionsumfang abdeckt. Den Probanden hat weiterhin die physikalische Simulation für das Layouting des Prototypen sowie die Möglichkeit, Einfluss darauf zu nehmen, gefallen.

9.1 Bewertung der Ergebnisse

Die gewonnenen Erkenntnisse bezüglich der Visualisierung von Massendaten beschränken sich weitestgehend auf das Layouting sowie den generellen Rendering-Prozess. Weiterhin wurden verschiedene Methoden identifiziert, mit deinen ein Traversieren durch

den Graph möglich ist. Da dieses Ergebnis zum Großteil auf bestehender Literatur aufbaut, ist mit hoher Wahrscheinlichkeit davon auszugehen, dass die daraus abgeleiteten fachlichen Features die tatsächlichen Anforderungen, die an so eine Anwendung gestellt werden, abdecken. Dies bestätigte sich teilweise duch die Validierung mithilfe der UX-Tests, allerdings haben manche Probanden auch noch weitere Features vermisst. Es ist also davon auszugehen, dass die erarbeiteten Features zwar alle relevant sind, aber noch nicht ausreichen und dass eine tiefere Arbeit in Kooperation mit Fachexperten notwendig ist, um den tatsächlichen Bedarf noch besser abdecken zu können.
Die erarbeitete Architektur bietet eine solide Grundlage, um eine komplexe Analysesoftware, so wie sie aus den Anforderungen hervor geht, zu implementieren. Dies zeigt sich vor Allem darin, dass es problemlos möglich war, eine detaillierte Implementierungsplanung für sämtliche identifizierte Features mithilfe der Architektur zu erstellen. Weiterhin konnten im Prototypen auch einige Architekturprinzipien validiert werden.
Das Zusammenspiel von vue.js und d3.js im Speziellen ist durch die Implementierung als ungenügend zu bewerten. Vermutlich wäre es besser, das Netzwerk komplett mit d3.js zu rendern und nicht vue.js in den Rendering-Prozess zu integrieren, auch wenn dies die Vorteile von vue.js wie Reactivity und State-Management für diesen speziellen Teil der Anwendung nicht nutzbar macht. Allerdings steht die Performance im Vordergrund und die ist leider mit vue.js zu stark eingeschränkt.

9.2 Ausblick

Auch in der Zukunft werden jedes Jahr unvorstellbare Mengen an Daten produziert werden, die ein großes Potential für Analysen der zu Grunde liegenden sozialen Netzwerke bieten.
Diese Arbeit hat das Fundament für Datenanalysen mithilfe von Twitter-Daten gelegt. Anhand der erarbeiteten Architektur und der verschiedenen Prozesse zum Datenimport, der Datenverarbeitung sowie der Analyse auf Basis des asynchronen Job-Frameworks ist ein Implementierungsleitfaden für spätere Implementierungen gegeben. Gerade durch die Erweiterbarkeit dieser Prozesse ist es problemlos möglich, weitere Analysekomponenten, wie beispielsweise eine Sentimentanalyse, einfach in den bestehenden Gesamtprozess zu integrieren.

Literatur

[AWS20] AWS. *Überblick Multipart Upload*. 2020. URL: https://docs.aws.amazon.com/de_de/AmazonS3/latest/dev/mpuoverview.html (besucht am 04. 01. 2020).

[BH86] Josh Barnes und Piet Hut. "A hierarchical O(N log N) force-calculation algorithm". In: *Nature* 324.6096 (Dez. 1986), S. 446–449. ISSN: 1476-4687. URL: https://doi.org/10.1038/324446a0.

[Clo] Google Cloud. *Nutzungsbasierte Preisgestaltung*. URL: https://cloud.google.com/maps-platform/pricing/ (besucht am 03. 12. 2019).

[Cru11] Alex Cruikshank. *Interactivity in HTML5 Canvas Visualizations*. 2011. URL: https://blog.carbonfive.com/2011/04/11/interactivity-in-html5-canvas-visualizations/ (besucht am 13. 01. 2020).

[d320] d3. *d3-force*. 2020. URL: https://github.com/d3/d3-force (besucht am 01. 01. 2020).

[DB-20] DB-Engines. *DB-Engines Ranking of Graph DBMS*. 2020. URL: https://db-engines.com/en/ranking/graph+dbms (besucht am 02. 01. 2020).

[Die17] Jean Diederich. "The API Economy. Why public APIs are so important". In: (2017). URL: https://www.wavestone.com/app/uploads/2017/10/Api-Economy-2017.pdf (besucht am 03. 12. 2019).

[Edi+10] D. Ediger u. a. "Massive Social Network Analysis: Mining Twitter for Social Good". In: *2010 39th International Conference on Parallel Processing*. Sep. 2010, S. 583–593. DOI: 10.1109/ICPP.2010.66.

[Elm+08] N. Elmqvist u. a. "ZAME: Interactive Large-Scale Graph Visualization". In: *2008 IEEE Pacific Visualization Symposium*. März 2008, S. 215–222. DOI: 10.1109/PACIFICVIS.2008.4475479.

[FR91] Thomas M. J. Fruchterman und Edward M. Reingold. "Graph Drawing by Force-directed Placement". In: *Software-Practice and Experience* 21 (1991).

[Fre04] Linton C. Freeman. "The Development of Social Network Analysis". In: (2004).

[Gri13] Ilya Grigorik. *High Performance Browser Networking*. 2013. URL: https://www.hpbn.co/http2 (besucht am 03. 01. 2020).

[Gyö+15] Cornelia Györödi u. a. "A Comparative Study: MongoDB vs MySQL". In: (2015).

[Jac+14] Mathieu Jacomy u. a. "ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software". In: *PLoS One* (2014).

[Jac19] Cam Jackson. *Micro Frontends*. 2019. URL: https://martinfowler.com/articles/micro-frontends.html (besucht am 28. 09. 2019).

[Jor+96] P. W. Jordan u. a., Hrsg. *SUS-A quick and dirty usability scale*. 1996. Kap. 21, S. 189–194.

[KST09] Stefan Kühl, Petra Strodtholz und Andreas Taffertshofer, Hrsg. *Handbuch Methoden der Organisationsforschung*. VS Verlag für Sozialwissenschaften, 2009. Kap. Netzwerkanalyse, S. 668–695. DOI: 10.1007/978-3-531-91570-8.

[Kur17] Jürgen Kuri. *Twitter verdoppelt maximale Länge der Tweets auf 280 Zeichen*. 2017. URL: https://www.heise.de/newsticker/meldung/Twitter-verdoppelt-maximale-Laenge-der-Tweets-auf-280-Zeichen-3883047.html (besucht am 01. 01. 2020).

[Kwa+10] Haewoon Kwak u. a. "What is Twitter, a Social Network or a News Media?" In: (2010).

[LMP83] Edward Laumann, Peter Marsden und David Prensky. "The Boundary Specification Problem in Network Analysis". In: *Applied Network Analysis: A Methodological Introduction* 61 (Jan. 1983).

[MW10] Peter V. Marsden und James D. Wright, Hrsg. *Question and Questionnaire Design*. 2010. Kap. 9, S. 263–313.

[MW11] Alexandra Marin und Barry Wellman. *The SAGE Handbook of Social Network Analysis*. Hrsg. von John Scott und Peter J. Carrington. 2011. Kap. 2, S. 11–22.

[npm20a] npm. *npm-d3*. 2020. URL: https://www.npmjs.com/package/d3 (besucht am 06. 01. 2020).

[npm20b] npm. *npm-sigma*. 2020. URL: https://www.npmjs.com/package/sigma (besucht am 06. 01. 2020).

[PFW00] Greg Parker, Glenn Franck und Cloin Ware. "Visualization of Large Nested Graphs in 3D: Navigation and Interaction". In: (2000).

[Pin18] Grace Pinegar. *What is Twitter? A Beginner's Guide*. 2018. URL: https://learn.g2.com/what-is-twitter (besucht am 30. 12. 2019).

[Smu09] Boris Smus. *Performance of canvas vs SVG*. 2009. URL: https://smus.com/canvas-vs-svg-performance/ (besucht am 13. 01. 2020).

[Sta18] Statista. *Global data center IP traffic from 2012 to 2021, by data center type (in exabytes per year)*. 2018. URL: https://www.statista.com/statistics/227268/global-data-center-ip-traffic-growth-by-data-center-type/ (besucht am 03. 12. 2019).

[Tel20] Telekom. *MagentaZuhause Geschwindigkeiten*. 2020. URL: https://www.telekom.de/zuhause/netz/geschwindigkeiten (besucht am 09. 01. 2020).

[Twia] Twitter. *Connecting to a Streaming Endpoint*. URL: https://developer.twitter.com/en/docs/tweets/filter-realtime/guides/connecting (besucht am 27. 11. 2019).

[Twib] Twitter. *Nutzung von Twitter*. URL: https://help.twitter.com/de/using-twitter (besucht am 22. 12. 2019).

10 Anhang

10.1 Fragebogen zur Validierung des Prototypen

Befragung zur Software Tweetalyzer

Szenario:

Ihre Aufgabe ist die Auswertung der politischen Werbung in den sozialen Medien für ein unabhängiges Institut. Speziell wollen Sie herausfinden, welche Arten von politischer Werbung von den verschiedenen Parteien genutzt werden und wie viele Menschen mit diesen Inhalten interagieren.

Als Datengrundlage dient hierbei ein Datensatz an Twitter-Daten, den Sie über die öffentliche Twitter-API abgerufen haben. Die Daten liegen in absoluter Rohform vor, identisch zum Format, dass von der Twitter-API geliefert wird.

Nun wollen Sie sich zunächst einen Überblick über die Daten verschaffen. Bei dieser Aufgabe soll Sie der Tweetalyzer unterstützen. Hierfür wurden die Daten nach Ihren Kriterien bereinigt, angereichert und in eine Graphdatenbank importiert. Der Tweetalyzer soll nun die Arbeit mit der Graphdatenbank vereinfachen. Weiterhin soll er eine Visualisierung des Datensatzes anbieten, über den sich intuitiv Zusammenhänge erschließbar machen.

Befragung:

Frage	Starker Widerspruch	Schwacher Widerspruch	Neutral	Schwache Zustimmung	Starke Zustimmung
Das System bietet einen regelmäßigen Nutzen für meine Arbeit					
Das System ist unnötig komplex					
Das System erscheint mir als einfach zu benutzen					
Zur Benutzung des Systems bräuchte ich technischen Support					
Die verschiedenen Funktionen des Systems sind gut integriert					
Das System beinhaltet viele Inkonsistenzen					
Die Arbeit mit dem System ist leicht zu erlernen					
Ich empfinde die Bedienung des Systems als umständlich					
Ich habe mich bei der Bedienung selbstbewusst und sicher gefühlt					
Ich musste eine Menge lernen, bevor ich das System benutzen konnte					

Kommentare:

10.2 Ergebnisse der Validierung des Prototypen

Befragung zur Software Tweetalyzer

Szenario:

Ihre Aufgabe ist die Auswertung der politischen Werbung in den sozialen Medien für ein unabhängiges Institut. Speziell wollen Sie herausfinden, welche Arten von politischer Werbung von den verschiedenen Parteien genutzt werden und wie viele Menschen mit diesen Inhalten interagieren.

Als Datengrundlage dient hierbei ein Datensatz an Twitter-Daten, den Sie über die öffentliche Twitter-API abgerufen haben. Die Daten liegen in absoluter Rohform vor, identisch zum Format, dass von der Twitter-API geliefert wird.

Nun wollen Sie sich zunächst einen Überblick über die Daten verschaffen. Bei dieser Aufgabe soll Sie der Tweetalyzer unterstützen. Hierfür wurden die Daten nach Ihren Kriterien bereinigt, angereichert und in eine Graphdatenbank importiert. Der Tweetalyzer soll nun die Arbeit mit der Graphdatenbank vereinfachen. Weiterhin soll er eine Visualisierung des Datensatzes anbieten, über den sich intuitiv Zusammenhänge erschließbar machen.

Befragung:

Frage	Starker Widerspruch	Schwacher Widerspruch	Neutral	Schwache Zustimmung	Starke Zustimmung
Das System bietet einen regelmäßigen Nutzen für meine Arbeit				x	
Das System ist unnötig komplex		x			
Das System erscheint mir als einfach zu benutzen				x	
Zur Benutzung des Systems bräuchte ich technischen Support	x				
Die verschiedenen Funktionen des Systems sind gut integriert			x		
Das System beinhaltet viele Inkonsistenzen		x			
Die Arbeit mit dem System ist leicht zu erlernen					x
Ich empfinde die Bedienung des Systems als umständlich		x			
Ich habe mich bei der Bedienung selbstbewusst und sicher gefühlt					x
Ich musste eine Menge lernen, bevor ich das System benutzen konnte	x				

Kommentare:

Cooler Prototyp!

Home-Seite wirkt sinnlos

Befragung zur Software Tweetalyzer

Szenario:

Ihre Aufgabe ist die Auswertung der politischen Werbung in den sozialen Medien für ein unabhängiges Institut. Speziell wollen Sie herausfinden, welche Arten von politischer Werbung von den verschiedenen Parteien genutzt werden und wie viele Menschen mit diesen Inhalten interagieren.

Als Datengrundlage dient hierbei ein Datensatz an Twitter-Daten, den Sie über die öffentliche Twitter-API abgerufen haben. Die Daten liegen in absoluter Rohform vor, identisch zum Format, dass von der Twitter-API geliefert wird.

Nun wollen Sie sich zunächst einen Überblick über die Daten verschaffen. Bei dieser Aufgabe soll Sie der Tweetalyzer unterstützen. Hierfür wurden die Daten nach Ihren Kriterien bereinigt, angereichert und in eine Graphdatenbank importiert. Der Tweetalyzer soll nun die Arbeit mit der Graphdatenbank vereinfachen. Weiterhin soll er eine Visualisierung des Datensatzes anbieten, über den sich intuitiv Zusammenhänge erschließbar machen.

Befragung:

Frage	Starker Widerspruch	Schwacher Widerspruch	Neutral	Schwache Zustimmung	Starke Zustimmung
Das System bietet einen regelmäßigen Nutzen für meine Arbeit					x
Das System ist unnötig komplex		x			
Das System erscheint mir als einfach zu benutzen					x
Zur Benutzung des Systems bräuchte ich technischen Support	x				
Die verschiedenen Funktionen des Systems sind gut integriert					x
Das System beinhaltet viele Inkonsistenzen		x			
Die Arbeit mit dem System ist leicht zu erlernen					x
Ich empfinde die Bedienung des Systems als umständlich	x				
Ich habe mich bei der Bedienung selbstbewusst und sicher gefühlt					x
Ich musste eine Menge lernen, bevor ich das System benutzen konnte	x				

Kommentare:

Ist nur ein Prototyp, könnte teilweise aber noch etwas stimmiger/ ästhetischer aussehen. Gerade die gewählten Farben im Netzwerk sind grausam

Warum nur so ein kleiner Teil des Netzwerks sichtbar?

Befragung zur Software Tweetalyzer

Szenario:

Ihre Aufgabe ist die Auswertung der politischen Werbung in den sozialen Medien für ein unabhängiges Institut. Speziell wollen Sie herausfinden, welche Arten von politischer Werbung von den verschiedenen Parteien genutzt werden und wie viele Menschen mit diesen Inhalten interagieren.

Als Datengrundlage dient hierbei ein Datensatz an Twitter-Daten, den Sie über die öffentliche Twitter-API abgerufen haben. Die Daten liegen in absoluter Rohform vor, identisch zum Format, dass von der Twitter-API geliefert wird.

Nun wollen Sie sich zunächst einen Überblick über die Daten verschaffen. Bei dieser Aufgabe soll Sie der Tweetalyzer unterstützen. Hierfür wurden die Daten nach Ihren Kriterien bereinigt, angereichert und in eine Graphdatenbank importiert. Der Tweetalyzer soll nun die Arbeit mit der Graphdatenbank vereinfachen. Weiterhin soll er eine Visualisierung des Datensatzes anbieten, über den sich intuitiv Zusammenhänge erschließbar machen.

Befragung:

Frage	Starker Widerspruch	Schwacher Widerspruch	Neutral	Schwache Zustimmung	Starke Zustimmung
Das System bietet einen regelmäßigen Nutzen für meine Arbeit			x		
Das System ist unnötig komplex	x				
Das System erscheint mir als einfach zu benutzen				x	
Zur Benutzung des Systems bräuchte ich technischen Support	x				
Die verschiedenen Funktionen des Systems sind gut integriert			x		
Das System beinhaltet viele Inkonsistenzen		x			
Die Arbeit mit dem System ist leicht zu erlernen					x
Ich empfinde die Bedienung des Systems als umständlich	x				
Ich habe mich bei der Bedienung selbstbewusst und sicher gefühlt				x	
Ich musste eine Menge lernen, bevor ich das System benutzen konnte		x			

Kommentare:

Funktionsumfang reicht für Beantwortung des Szenarios nicht aus, z.b. keinerlei Infos über Interaktionshäufigkeiten, Dichte etc. – für einen Überblick aber ok!

Befragung zur Software Tweetalyzer

Szenario:

Ihre Aufgabe ist die Auswertung der politischen Werbung in den sozialen Medien für ein unabhängiges Institut. Speziell wollen Sie herausfinden, welche Arten von politischer Werbung von den verschiedenen Parteien genutzt werden und wie viele Menschen mit diesen Inhalten interagieren.

Als Datengrundlage dient hierbei ein Datensatz an Twitter-Daten, den Sie über die öffentliche Twitter-API abgerufen haben. Die Daten liegen in absoluter Rohform vor, identisch zum Format, dass von der Twitter-API geliefert wird.

Nun wollen Sie sich zunächst einen Überblick über die Daten verschaffen. Bei dieser Aufgabe soll Sie der Tweetalyzer unterstützen. Hierfür wurden die Daten nach Ihren Kriterien bereinigt, angereichert und in eine Graphdatenbank importiert. Der Tweetalyzer soll nun die Arbeit mit der Graphdatenbank vereinfachen. Weiterhin soll er eine Visualisierung des Datensatzes anbieten, über den sich intuitiv Zusammenhänge erschließbar machen.

Befragung:

Frage	Starker Widerspruch	Schwacher Widerspruch	Neutral	Schwache Zustimmung	Starke Zustimmung
Das System bietet einen regelmäßigen Nutzen für meine Arbeit				x	
Das System ist unnötig komplex	x				
Das System erscheint mir als einfach zu benutzen				x	
Zur Benutzung des Systems bräuchte ich technischen Support		x			
Die verschiedenen Funktionen des Systems sind gut integriert					x
Das System beinhaltet viele Inkonsistenzen		x			
Die Arbeit mit dem System ist leicht zu erlernen					x
Ich empfinde die Bedienung des Systems als umständlich				x	
Ich habe mich bei der Bedienung selbstbewusst und sicher gefühlt			x		
Ich musste eine Menge lernen, bevor ich das System benutzen konnte	x				

Kommentare:

Interaktionen mit dem Graphen teilweise inkonsitstent, beispielsweise bleibt Knotenpunkt beim Verschieben manchmal am Cursor hängen

Physikalische Simulation ist total cool!

Befragung zur Software Tweetalyzer

Szenario:

Ihre Aufgabe ist die Auswertung der politischen Werbung in den sozialen Medien für ein unabhängiges Institut. Speziell wollen Sie herausfinden, welche Arten von politischer Werbung von den verschiedenen Parteien genutzt werden und wie viele Menschen mit diesen Inhalten interagieren.

Als Datengrundlage dient hierbei ein Datensatz an Twitter-Daten, den Sie über die öffentliche Twitter-API abgerufen haben. Die Daten liegen in absoluter Rohform vor, identisch zum Format, dass von der Twitter-API geliefert wird.

Nun wollen Sie sich zunächst einen Überblick über die Daten verschaffen. Bei dieser Aufgabe soll Sie der Tweetalyzer unterstützen. Hierfür wurden die Daten nach Ihren Kriterien bereinigt, angereichert und in eine Graphdatenbank importiert. Der Tweetalyzer soll nun die Arbeit mit der Graphdatenbank vereinfachen. Weiterhin soll er eine Visualisierung des Datensatzes anbieten, über den sich intuitiv Zusammenhänge erschließbar machen.

Befragung:

Frage	Starker Widerspruch	Schwacher Widerspruch	Neutral	Schwache Zustimmung	Starke Zustimmung
Das System bietet einen regelmäßigen Nutzen für meine Arbeit					x
Das System ist unnötig komplex	x				
Das System erscheint mir als einfach zu benutzen				x	
Zur Benutzung des Systems bräuchte ich technischen Support	x				
Die verschiedenen Funktionen des Systems sind gut integriert					x
Das System beinhaltet viele Inkonsistenzen	x				
Die Arbeit mit dem System ist leicht zu erlernen					x
Ich empfinde die Bedienung des Systems als umständlich	x				
Ich habe mich bei der Bedienung selbstbewusst und sicher gefühlt				x	
Ich musste eine Menge lernen, bevor ich das System benutzen konnte	x				

Kommentare:

Die Simulation anhand physikalischer Merkmale fühlt sich gut an

DANKE dass wir Data Scientists Einfluss auf das Layout nehmen dürfen – häufig ist die eingesetzte Software da deutlich restrktiver, was die Arbeit sehr mühsam macht

Befragung zur Software Tweetalyzer

Szenario:

Ihre Aufgabe ist die Auswertung der politischen Werbung in den sozialen Medien für ein unabhängiges Institut. Speziell wollen Sie herausfinden, welche Arten von politischer Werbung von den verschiedenen Parteien genutzt werden und wie viele Menschen mit diesen Inhalten interagieren.

Als Datengrundlage dient hierbei ein Datensatz an Twitter-Daten, den Sie über die öffentliche Twitter-API abgerufen haben. Die Daten liegen in absoluter Rohform vor, identisch zum Format, dass von der Twitter-API geliefert wird.

Nun wollen Sie sich zunächst einen Überblick über die Daten verschaffen. Bei dieser Aufgabe soll Sie der Tweetalyzer unterstützen. Hierfür wurden die Daten nach Ihren Kriterien bereinigt, angereichert und in eine Graphdatenbank importiert. Der Tweetalyzer soll nun die Arbeit mit der Graphdatenbank vereinfachen. Weiterhin soll er eine Visualisierung des Datensatzes anbieten, über den sich intuitiv Zusammenhänge erschließbar machen.

Befragung:

Frage	Starker Widerspruch	Schwacher Widerspruch	Neutral	Schwache Zustimmung	Starke Zustimmung
Das System bietet einen regelmäßigen Nutzen für meine Arbeit		x			
Das System ist unnötig komplex	x				
Das System erscheint mir als einfach zu benutzen			x		
Zur Benutzung des Systems bräuchte ich technischen Support	x				
Die verschiedenen Funktionen des Systems sind gut integriert			x		
Das System beinhaltet viele Inkonsistenzen		x			
Die Arbeit mit dem System ist leicht zu erlernen					x
Ich empfinde die Bedienung des Systems als umständlich	x				
Ich habe mich bei der Bedienung selbstbewusst und sicher gefühlt					x
Ich musste eine Menge lernen, bevor ich das System benutzen konnte	x				

Kommentare:

Fortschrittsanzeige beim Datenimport ist super! Wir müssen häufig die Entwickler fragen, wie der Zustand bei verschiedenen Importprozessen in BRAIN und Co. ist…

Der Nutzen der Home-Seite erschließt sich mir nicht – warum nicht gleich mit dem Netzwerk einsteigen?

Befragung zur Software Tweetalyzer

Szenario:

Ihre Aufgabe ist die Auswertung der politischen Werbung in den sozialen Medien für ein unabhängiges Institut. Speziell wollen Sie herausfinden, welche Arten von politischer Werbung von den verschiedenen Parteien genutzt werden und wie viele Menschen mit diesen Inhalten interagieren.

Als Datengrundlage dient hierbei ein Datensatz an Twitter-Daten, den Sie über die öffentliche Twitter-API abgerufen haben. Die Daten liegen in absoluter Rohform vor, identisch zum Format, dass von der Twitter-API geliefert wird.

Nun wollen Sie sich zunächst einen Überblick über die Daten verschaffen. Bei dieser Aufgabe soll Sie der Tweetalyzer unterstützen. Hierfür wurden die Daten nach Ihren Kriterien bereinigt, angereichert und in eine Graphdatenbank importiert. Der Tweetalyzer soll nun die Arbeit mit der Graphdatenbank vereinfachen. Weiterhin soll er eine Visualisierung des Datensatzes anbieten, über den sich intuitiv Zusammenhänge erschließbar machen.

Befragung:

Frage	Starker Widerspruch	Schwacher Widerspruch	Neutral	Schwache Zustimmung	Starke Zustimmung
Das System bietet einen regelmäßigen Nutzen für meine Arbeit			x		
Das System ist unnötig komplex	x				
Das System erscheint mir als einfach zu benutzen				x	
Zur Benutzung des Systems bräuchte ich technischen Support	x				
Die verschiedenen Funktionen des Systems sind gut integriert					x
Das System beinhaltet viele Inkonsistenzen	x				
Die Arbeit mit dem System ist leicht zu erlernen					x
Ich empfinde die Bedienung des Systems als umständlich	x				
Ich habe mich bei der Bedienung selbstbewusst und sicher gefühlt				x	
Ich musste eine Menge lernen, bevor ich das System benutzen konnte	x				

Kommentare:

Home-Seite wirkt sinnlos und deplaziert

Performance bricht bei 500+ Knotenpunkten deutlich ein

Befragung zur Software Tweetalyzer

Szenario:

Ihre Aufgabe ist die Auswertung der politischen Werbung in den sozialen Medien für ein unabhängiges Institut. Speziell wollen Sie herausfinden, welche Arten von politischer Werbung von den verschiedenen Parteien genutzt werden und wie viele Menschen mit diesen Inhalten interagieren.

Als Datengrundlage dient hierbei ein Datensatz an Twitter-Daten, den Sie über die öffentliche Twitter-API abgerufen haben. Die Daten liegen in absoluter Rohform vor, identisch zum Format, dass von der Twitter-API geliefert wird.

Nun wollen Sie sich zunächst einen Überblick über die Daten verschaffen. Bei dieser Aufgabe soll Sie der Tweetalyzer unterstützen. Hierfür wurden die Daten nach Ihren Kriterien bereinigt, angereichert und in eine Graphdatenbank importiert. Der Tweetalyzer soll nun die Arbeit mit der Graphdatenbank vereinfachen. Weiterhin soll er eine Visualisierung des Datensatzes anbieten, über den sich intuitiv Zusammenhänge erschließbar machen.

Befragung:

Frage	Starker Widerspruch	Schwacher Widerspruch	Neutral	Schwache Zustimmung	Starke Zustimmung
Das System bietet einen regelmäßigen Nutzen für meine Arbeit		X			
Das System ist unnötig komplex	x				
Das System erscheint mir als einfach zu benutzen					x
Zur Benutzung des Systems bräuchte ich technischen Support					x
Die verschiedenen Funktionen des Systems sind gut integriert					x
Das System beinhaltet viele Inkonsistenzen		x			
Die Arbeit mit dem System ist leicht zu erlernen			x		
Ich empfinde die Bedienung des Systems als umständlich		x			
Ich habe mich bei der Bedienung selbstbewusst und sicher gefühlt					x
Ich musste eine Menge lernen, bevor ich das System benutzen konnte		x			

Kommentare:

Zu starke Einschränkung! Warum nicht direkt Cypher-Queries eingeben? Wir nutzen doch auch neo4j! Wie kann ich die Daten modifizieren? Daher bräuchte ich wieder technischen Support durch Software Devs, die die Software dann so konfugirieren, wie ich es gerade benötige

Home-Seite kann weg