Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Projektbericht
für das Praktikum base.camp

Wikipedia People Graph

Github: https://github.com/dasmerlon/wikipedia-people-graph

Demo: http://basecamp-demos.informatik.uni-hamburg.de:8080/peoplegraph

vorgelegt von

Murad Caliskan, Merle Hoffmann,
Henning Möllers, Christian Wolff

eingereicht am 16. April 2021

Betreuer: Eugen Ruppert

Inhaltsverzeichnis

1 **Einführung** (Henning)

Im Rahmen des base.camp Praktikums des Wintersemesters 2020/2021 wurden die verschiedenen Teilnehmergruppen vor Herausforderungen aus dem Themengebiet “Big Data” gestellt. Dabei konnten sich die Gruppen zwischen verschiedenen Datensätzen und Datenquellen entscheiden. Unsere Gruppe interessierte sich für eine Verarbeitung der gesamten Wikipedia-Daten und insbesondere für die darin enthaltenen Daten zu Personen. Die sogenannten Wikimedia-Dumps, die unter dumps.wikimedia.org abgerufen werden können, enthalten die Daten aller Wikipedia-Artikel einer Sprache.

Unsere Absicht war es einerseits klassische personenbezogene Daten, wie Beruf, Geburtsdatum und Todesdatum aus den Wikipedia-Daten zu extrahieren. Diese Daten wollten wir dann entsprechend visualisieren und die Informationen so für Nutzer zugänglicher und anschaulicher machen. Für diesen Zweck bot sich die Erstellung eines interaktiven Zeistrahls an, der die Lebenszeiträume von verschiedenen Personen abbildet und Zusatzinformationen anbietet.

Neben den klassischen Personendaten interessierte sich unsere Gruppe insbesondere für die Beziehungen zwischen den verschiedenen Personen aus Wikipedia. Die in den Wikipedia-Artikeln aufgeführten Querverweise auf die Artikel anderer Personen boten sich hier als Datenquelle an. Wir wollten die Netzwerke von Personen, die sich aus der Gesamtheit dieser Querverweise ergeben würden, als solche visualisieren und damit dem Nutzer deutlich einfacher zugänglich machen.

Unsere Gruppe verfolgte also von Anfang an zwei verschiedene Ziele für eine Visualisierung (Zeitstrahl und Netzwerkgraph). Da in beiden Darstellungen allerdings jegliche Daten einen Personenbezug hatten, würde sich ein großes Potenzial bieten, die Daten beider Visualisierungen miteinander zu verknüpfen und somit einen noch größeren Mehrwert für den Nutzer gegenüber den herkömmlichen Wikipedia-Artikeln zu generieren.

2 Organisation (Murad)

Nach der Gruppeneinteilung haben wir uns schnell abgesprochen und in einem Discord-Server getroffen. Discord ist ein Onlinedienst für Instant Messaging, Chat, Sprachkonferenzen und Videokonferenzen. Dieser war anfangs für Computerspieler gedacht, hat sich aber inzwischen auch für andere Bereiche als nützlich erwiesen. Da wir uns untereinander nicht kannten, haben wir uns erst einmal kennengelernt und zu unserem Thema gebrainstormt. Discord war dafür eine sehr gute Wahl, um miteinander reden zu können und auch mal seinen Bildschirm zu teilen, um aufgekommene Probleme besser zusammen beheben zu können. Discord wurde für uns zur Plattform zum sprachlichen Austausch in abgemachten Gruppentreffen oder gemeinsamer Arbeit am Projekt. Über den Instant Messaging Dienst Whatsapp blieben wir alle untereinander erreichbar und haben uns für Gruppentreffen auf dem Discord verabredet. Um uns gegenseitig mit dem Fortschritt der Programmierung auf dem Laufenden zu halten, haben wir uns GitHub zu Hilfe genommen. GitHub bietet den Nutzern die Möglichkeit ihre Quelltexte auf einer „Datenbank“ zu speichern, den sogenannten Repositories. Alle Nutzer, die zu dem GitHub-Projekt eingeladen sind und somit Zugriff haben, können die aktuellsten Version des Quelltextes einsehen und an dieser arbeiten. Wenn ein Repository öffentlich ist, kann jeder den Quelltext einsehen und herunterladen. Da die Bedienung auch über die Weboberfläche von GitHub möglich ist, ist die Bedienung recht anfängerfreundlich.

2.1 Designentscheidungen

Beim Design wurde uns die Ergebnisvisualisierung mittels einer Webseite vorgegeben. Da sich unser Projekt in einen Netzwerkgraphen und einen Zeitstrahl aufteilt, hatten wir beschlossen zwei Webseiten aufzubauen und zum Ende hin für die Nutzbarkeit zusammenzufügen. Nach ein paar Test-Templates für Webseiten sind wir auf ein geeignetes Design für den Zeitstrahl, sowie für den Netzwerkgraphen, gestoßen. Die Webseite des Netzwerkgraphen soll dem Nutzer veranschaulichen können, welche Beziehungen eine ausgewählte Person zu anderen Personen hat. Dabei bekommt der Nutzer selbst die Wahl, welche Person mit ihren Beziehungen veranschaulicht werden soll. Des Weiteren erweitert sich der Graph auch auf eine zweiten Ebene, die die Beziehungen der Personen anzeigt, die eine Beziehung zur ausgewählten Person haben. Der Zeitstrahl hingegen bietet dem Nutzer eine Visualisierung des Lebenszeitraumes einer oder mehrere Personen. Dabei kann man sehen, wann eine Person geboren und, falls sie nicht mehr lebt, auch gestorben ist. Um dem noch eine persönlichere Note zu geben, bekommt man bei der Auswahl einer Person auch einen Steckbrief mit einem Bild gezeigt, falls dieses vorhanden ist.

2.2 Herangehensweise

Das im Praktikum benutzte Hadoop-System war für alle in der Gruppe etwas neues, wodurch vorerst Bedarf bestand sich in das Thema einzulesen und mit Beispielprojekten zu arbeiten, um

einen Überblick über die Funktionsweise zu bekommen. Damit sich alle in Hadoop einfinden konnten, arbeiteten wir am Anfang alle zusammen am Programmcode für den MapReduce-Job. Wir haben verschiedene Herangehensweisen ausprobiert, um aus den Daten die für uns relevanten Informationen herauszufiltern und sind dabei auch oft auf Ausnahmefälle gestoßen, wie z.B. das eine fiktive Person als lebende Person aufgefasst wurde. Nachdem ein Großteil der Logik hinter dem Datenfilterungsjob fertig war, verteilten wir unseren Aufgabenbereich, um zeitgleich auch mit unseren anderen Baustellen, insbesondere der Webseite, beginnen zu können.

3 Implementierung

3.1 Architektur (Christian, Murad)

Grundlegende Architektur

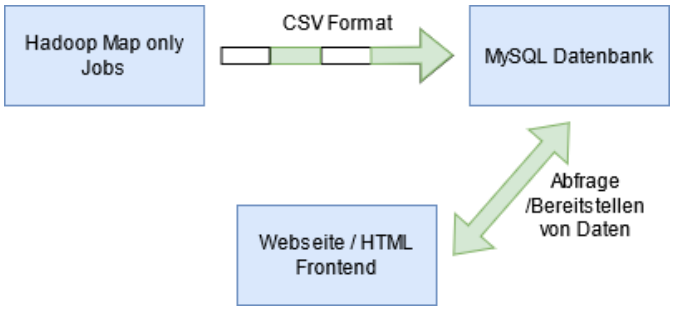

Abbildung 3.1: Veranschaulichung der Architektur des Projektes.

Wie in Abbildung 3.1 ersichtlich, setzt sich unser Projekt aus drei Kernelementen zusammen, den Hadoop Map-Only Jobs aus dem Hadoop File System (HDFS), dem MySQL-Server und dem HTML Frontend bzw. Webserver. Die Hadoop Map-Only Jobs unterteilen sich in insgesamt vier unterschiedliche Jobs, deren Funktionsweise in Abschnitt 3.2.2 genauer erläutert werden. An dieser Stelle sei nur erwähnt, dass diese vier Jobs die entscheidende Arbeit leisten, um aus allen Wikipedia-Artikeln die Personen, Beziehungen und wichtigen Informationen zu extrahieren und damit die Datenbasis für unser Projekt liefern. Das Ausgabeformat nach dem Durchlaufen der unterschiedlichen Map-Jobs sind zwei CSV-Dateien, die einmalig manuell in die MySQL-Datenbank geladen werden. Zur Aktualisierung der Daten ist ein erneutes Herunterladen von neuen Wikipedia Dumps möglich. Diese müssen nur mit unseren Map-Only Jobs über das Hadoop File System entsprechend verarbeitet und dann auf den MySQL-Server geladen werden.

In der MySQL-Datenbank sind zwei Tabellen enthalten mit deren Hilfe die Informationen zur grafischen Aufbereitung der Personen, ihrer persönlichen Daten und ihrer Beziehung zueinander für den Zeitstahl und den Netzwerkgraphen auf Abruf bereitgestellt werden. Der Abruf der Daten erfolgt durch das Frontend bzw. die Webseite. Diese stellt ihre erste Anfrage an die Datenbank schon beim ersten Aufruf der Webseite. Hier werden gezielt Personen mit “Einstein”

im Namen abgefragt, um dem Nutzer ein Beispiel zu zeigen, wie die Webseite funktioniert. Durch Klicken des Nutzers auf einen Namen oder durch das Ausfüllen von einem oder mehreren Suchfeld(ern), erfolgen weitere Abfragen an die Datenbank.

So entsteht durch Nutzerinteraktion eine Abfolge von Datenanforderungen und Datenbereitstellungen zwischen dem MySQL-Server und der Webseite. Sobald die Anfragen abgearbeitet wurden und keine neuen Nutzerinteraktionen ausgeführt werden, ruht die gegenseitige Kommunikation zwischen der Datenbank und dem Frontend.

Architektur der Webseite

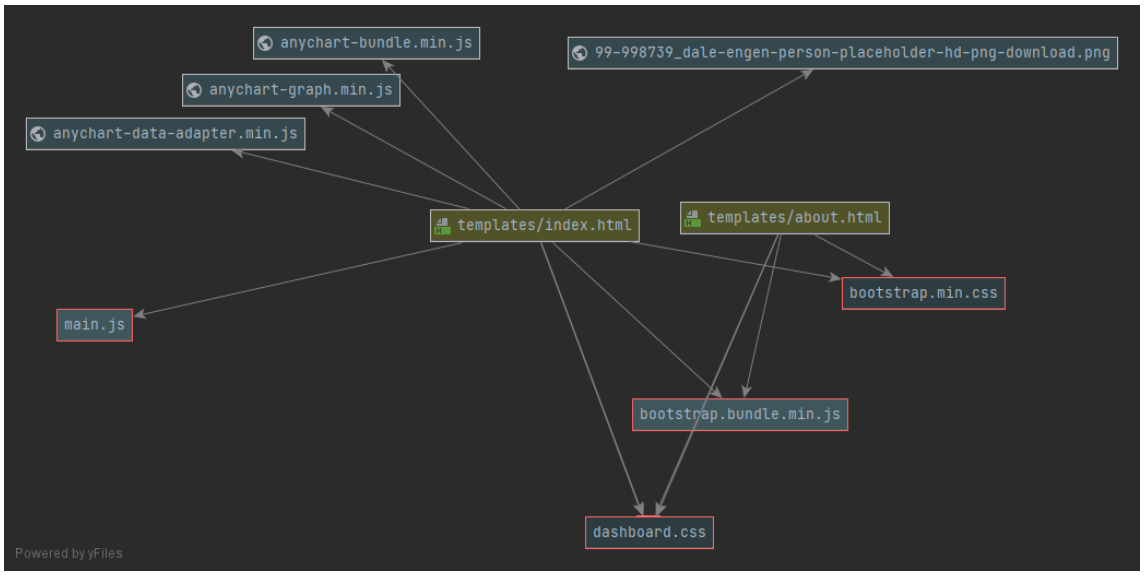

Abbildung 3.2: Architektur der Webseite als UML

Die Webseite wird auf einem Tomcat-Server des base.camps der Universität Hamburg bereitgestellt. Für den Tomcat-Server selbst erfolgte keine Einrichtung. Das Projekt musste lediglich als WAR-Datei bereitgestellt und hochgeladen werden. Der Abbildung 3.2 kann der Aufbau der Webseite entnommen werden. Grundlegend basiert die Webseite auf dem Spring-Framework und ist eine Spring-Boot-Anwendung.

Die `index.html` und `about.html` stellen den Rahmen für die Webseite bereit. Beide basieren auf dem "dashboard"-Beispielprojekt aus Bootstrap 5.0.0. Während die `about.html` lediglich eine eigene Sub-Webseite mit einem Erklärtext des Projektes darstellt, ist die `index.html` der Grundpfeiler für den Aufbau der Webseite. Hier sind die Container zur Anzeige des Zeitstrahls und des Netzwerkgraphen definiert, welche über die AnyChart-Library bereitgestellt werden. Auch ist ein Platzhalterbild hinterlegt, falls zu einer Person kein Bild in der SQL-Tabelle hinterlegt ist oder das Bild nicht richtig angezeigt werden kann. Gesteuert werden die Inhalte für den Zeitstrahl und Netzwerkgraph über die `main.js`. In dieser sind die Funktionen zum Bauen der beiden Visualisierungen hinterlegt. Eine detaillierte Erklärung zum Zusammenwirken der einzelnen Elemente kann in Abschnitt 3.4 eingesehen werden.

3.2 Big-Data Analyse

3.2.1 **Daten scrapen vs. Wikipedia Dumps** (Murad)

Wikipedia wuchs über die Jahre um mehr als 6 Millionen verfasste englischsprachige Artikel, die nicht nur umfangreich sind, sondern auch unterschiedlicher kaum sein könnten. Bei unserem Praktikum wurde uns die Entscheidung selbst überlassen, wie wir die Rohdaten der Personenartikel erfassen wollten. Dabei tat sich die Wahl auf, die Wikipedia Dumps zu nutzen oder die Daten selber zu scrapen.

Daten selbst scrapen

Heutzutage gibt es viele fertige Tools mit denen man Daten und Artikel aus Wikipedia extrahieren kann. Zwar sind diese oft einfach zu bedienen, allerdings bieten sie leider keine Garantie auf Vollständigkeit, wenn man sie in einem so großen Umfang benutzen würde. Da wir in den Wikipedia-Artikeln von Personen keinen Anhaltspunkt gefunden haben, der direkt auf eine Person hinweist, bliebe uns nur die Möglichkeit alle Artikel zu extrahieren. Dieser Vorgang würde bei mehreren Millionen Artikeln mehrere Tage andauern und gegebenenfalls auch ein ganz anderes Problem herbeiführen, den DDOS Schutz. Große Webseiten, wie auch Wikipedia, schützen sich vor solchen Attacken von Anfragen, um einer Überlastung vorzubeugen. Durch den Umfang unserer Extraktion unter einer IP-Adresse wäre es naheliegend, das unsere Anfragen nach einiger Zeit blockiert werden.

Wikipedia Dumps

Wikipedia Dumps sind von Wikimedia zur Verfügung gestellte, komplette Backups ihrer Datenbank. Diese stehen jedem legal und jederzeit zur Benutzung frei. Diese Backups werden ein- bis zweimal im Monat aktualisiert und können somit als recht aktuell betrachtet werden. Die Daten stehen dann verpackt in einer XML-Datei zur Verfügung und beinhalten je nach Wahl bis zu allen Wikipedia-Artikeln in einer ausgewählten Sprache.

Nach Abwägung zwischen Vollständigkeit und benötigtem Aufwand zum Extrahieren der Personendaten, haben wir uns für die englischen Wikipedia Dumps entschieden. Hierbei standen uns die Daten, im Gegensatz zum Scraping-Ansatz, vollständig und sofort zum Download bereit.

3.2.2 **MapReduce** (Merle)

MapReduce ist ein Programmiermodell, bzw. ein Software-Framework, das nebenläufige Berechnungen von großen Datenmengen ermöglicht. Dabei wird die Verarbeitung normalerweise in zwei Phasen aufgeteilt. Die Map-Phase dient der Filterung und der generellen Verarbeitung der Daten, während die Reduce-Phase die Ergebnisse des sogenannten Mappers weiterverarbeitet und zusammenfasst. Ein MapReduce-Job teilt den Datensatz in unabhängige Chunks auf, die von den Mappern vollständig parallel verarbeitet werden. Anschließend werden die Ausgaben des Mappers sortiert und dem Reducer übergegeben. Durch die unabhängige Aufteilung lassen sich sehr große Datenmengen parallel auf Clustern von Standardhardware zuverlässig und

fehlertolerant verarbeiten. Die Ein- und Ausgabe des Jobs wird dabei in der Regel in einem Dateisystem gespeichert. In unserem Praktikum wurde dafür das Hadoop Distributed File System (HDFS) verwendet. Es handelt sich hierbei um ein verteiltes Dateisystem, welches über ein Hadoop-Cluster den Zugriff auf Daten ermöglicht.

MapReduce arbeitet ausschließlich mit Key-Value-Paaren. Die Eingabe in den MapReduce-Job wird demnach als eine Menge von Key-Value Paaren betrachtet und auch als Ausgabe wird eine Menge von Key-Value Paaren erzeugt. Da MapReduce immer nur mit zwei Werten arbeitet, kann es etwas schwieriger sein, komplexere Daten zu verarbeiten, wenn man beispielsweise mehr als zwei Daten innerhalb eines Jobs extrahieren möchte. Um dieses Problem zu umgehen, haben wir uns dazu entschieden, mehrere MapReduce-Jobs zu nutzen, die unterschiedliche Aufgaben übernehmen.

Eingabeformat der MapReduce-Jobs

Wie schon in Abschnitt 3.2.1 beschrieben, haben wir die Wikimedia Dumps für unsere Datenanalyse genutzt. Für das Projekt haben wir den Dump `enwiki-latest-pages-meta-current.xml` verwendet. Dies ist eine einzelne XML-Datei, die alle englischen Wikipedia-Artikel auf dem neusten Stand enthält. Zu der Zeit unseres Downloads war der letzte Stand von Februar 2021.

Wie zu Beginn dieses Abschnittes 3.2.2 bereits erwähnt, teilt ein MapReduce-Job den Datensatz in unabhängige Chunks auf. Es gibt verschiedene vordefinierte Eingabeformate, sogenannte `InputFormatClasses`, die festlegen in welche Chunks der Datensatz aufgeteilt werden soll. Beispielsweise werden Eingabedaten häufig in Zeilen aufgeteilt und verarbeitet, wenn es sich bei den Daten um Text handelt. Da diese Zeilen allerdings unabhängig voneinander verarbeitet werden, können in unserem Beispiel keine Rückschlüsse gezogen werden, welche Zeile aus welchem Wikipedia-Artikel stammt. Für dieses Problem haben wir uns zwei unterschiedliche Lösungsansätze überlegt. Eine Möglichkeit bestand darin, die Zeilen in der Map-Phase mit einem eindeutigen Key zu taggen, sodass zusammengehörige Zeilen den gleichen Key besitzen und in der Reduce-Phase wieder zusammengetragen werden könnten.

Die andere Idee, die wir letztendlich auch umgesetzt haben, beinhaltet das Aufteilen des Datensatzes in Wikipedia-Seiten, sodass jeder Mapper einen ganzen Wikipedia-Artikel erhält und verarbeiten kann. Da unser Datensatz im XML- bzw. HTML-Format vorliegt und dementsprechend Start- und Endtags beinhaltet, bot sich dieser Ansatz an. Als Vorlage für die neue `InputFormatClass` haben wir ein Projekt von Thomas P. Moyer verwendet, welches wir an unseren Anwendungsfall leicht angepasst haben. Dieses Eingabeformat ist ein `XmlInputFormat`, bei dem man einen Start- und Endtag festlegt, die den Start und das Ende eines Chunks definieren. In unserem Fall haben wir die Tags `<page>` und `</page>` gewählt, da diese die Wikipedia-Artikel umfassen.

Ein großer Vorteil dieser Lösung ist, dass wir uns nun die Reduce-Phasen in den nachfolgenden Jobs sparen können, da wir die Wikipedia-Artikel nicht aufsplitten und somit der Reducer auch nichts zusammentragen muss. Aus diesem Grund benötigen wir nur Map-Only Jobs, die alle das `XmlInputFormat` nutzen.

PersonArticleExtractor

Die Dateigröße des unverarbeiteten Wikimedia-Dumps beträgt 169.2 GB. Da uns aber nur ein kleiner Teil dieser Daten interessiert, entschieden wir uns zunächst einen `PersonArticleExtractor` zu bauen, der aus dem Dump alle Personenartikel herausfiltert und einen neuen Dump erstellt, der nur Personenartikel enthält. In diesem Job räumen wir auch gleichzeitig schon die Personenartikel etwas auf, indem wir die für uns uninteressanten Referenzen und Userkommentare entfernen und Sonderzeichen, die als HTML-Codierung enthalten sind, decodieren. Der Job ermöglicht es uns die eigentliche Datenverarbeitung auf einem wesentlich kleineren Datensatz durchzuführen und sorgt somit für eine deutlich geringere Laufzeit. Die Datenmenge betrug nach dem Durchlauf des Jobs nur noch 7.4 GB.

Wikipedia-Artikel lassen sich nicht durch Typen oder ähnliches unterscheiden. Die englischen Wikipedia-Artikel enthalten allerdings mehrere unsortierte Kategorien. Personenartikel besitzen in der Regel eine Geburtskategorie nach folgendem Schema: `[[Category:<Jahr> births]]`. Das `<Jahr>` kann dabei eine ein- bis vierstellige ganze Zahl sein, die zusätzlich ein `BC` enthalten kann. Diese Geburtskategorie haben wir genutzt, um Personenartikel zu identifizieren. Allerdings können Personen mit unbekanntem Geburtsjahr oder unvollständigem Artikel folglich nicht gefunden werden. Diesen Verlust haben wir in Kauf genommen, da die meisten und wichtigsten Personen ein Geburtsjahr eingetragen haben sollten und dies eine zuverlässige Methode ist, um Personenartikel zu erkennen.

Der Mapper des `PersonArticleExtractors` bekommt als Eingabe einen vollständigen Wikipedia-Artikel und prüft, ob es sich dabei um einen Personenartikel handelt. Dafür geht der Mapper den Artikel Zeile für Zeile durch und prüft, ob die Zeile die Geburtskategorie, welche durch einen Regex-Ausdruck beschrieben wird, enthält. Wenn der Mapper eine solche Zeile findet, entfernt er alle Referenzen und Userkommentare im Artikel. Die Userkommentare entsprechen dem Aufbau von HTML-Kommentaren: `<!-- Kommentar -->`. Die Referenzen haben leider keine einheitliche Syntax. Am häufigsten treten folgende Syntaxen auf: `<ref ...>Referenz</ref>`, `{{sfn... Referenz}}`, `{{efn... Referenz}}` oder `{{ref... Referenz}}`. Mithilfe von Regex-Ausdrücken werden diese Syntaxen gefunden und entfernt. Zum Schluss werden noch HTML-codierte Sonderzeichen decodiert. Dafür haben wir im Mapper eine Methode erstellt, die bestimmten HTML-Code durch das entsprechende Sonderzeichen ersetzt. Die Wikipedia-Artikel aus dem Dump wurden anscheinend zweifach codiert. Da die Codierungen selbst Und-Zeichen enthalten, muss der HTML-Decoder das Und-Zeichen doppelt decodieren.

Der verarbeitete Personenartikel wird anschließend als Output-Key zurückgegeben, wobei der Output-Value leer bleibt. Der Job erstellt als Ergebnis mehrere Output-Dateien. Diese haben wir anschließend im HDFS manuell zu einer XML-Datei zusammengefügt, damit wir die dadurch entstehende XML-Datei als Eingabedatei für alle weiteren Jobs verwenden konnten. Zum Zusammenfügen der Ausgabe bietet sich folgender Befehl an: `hadoop fs -getmerge \$OUTPUT/PersonArticle person_article.xml`.

PersonData

Der Map-Only Job `PersonData` erhält als Eingabe die Ausgabe von `PersonArticleExtractor`. Ein Mapper bekommt demnach immer einen ganzen Personenartikel. Diesen Artikel geht er Zeile für Zeile durch und extrahiert die von uns gesuchten Informationen zu der Person. Zunächst prüft der Mapper, ob der Titel den String "`user:`" enthält und überspringt diese

Artikel. Dies ist notwendig, da es sich bei diesen Artikeln um Sandbox-Seiten handelt. Sandbox-Seiten sind von Usern erstellte Testartikel. Besser wäre es gewesen, diese Seiten bereits im `PersonArticleExtractor` zu entfernen. Allerdings sind wir erst später auf dieses Problem gestoßen, weshalb wir aus zeitlichen Gründen die Sandbox-Seiten erst nachträglich im `PersonData` Job entfernt haben, anstatt den `PersonArticleExtractor` neu durchlaufen zu lassen.

Die Informationen zu den Personen werden in einer von uns festgelegten Reihenfolge gesammelt und mit einem speziellen eindeutigen Delimiter `>>>>` voneinander getrennt. Wenn zu einer Information nichts gefunden werden konnte, erhält sie den Eintrag `NONE`. Somit bleibt der Aufbau und die Reihenfolge bei allen Personen gleich, was uns später hilft die Daten einfach in die Datenbank zu laden. Die Ausgabe eines Mappers enthält folgende Informationen:
`TITLE, URL, SHORT_DESCRIPTION, IMAGE, NAME, BIRTH_NAME, BIRTH_DATE, BIRTH_PLACE, DEATH_DATE, DEATH_PLACE, DEATH_CAUSE, NATIONALITY, EDUCATION, KNOWN_FOR, OCCUPATION, ORDER, OFFICE, TERM_START, TERM_END, PARTY.`

Nicht alle Informationen haben wir letztendlich auch in unsere Web-App eingebunden. Im Folgenden werden wir nur auf die relevantesten Daten eingehen. Für eine genauere Erklärung der Datenextraktion siehe den Quellcode des PersonDataMappers ein.

Der Titel eines Personenartikels ist immer der Name der Person mit gegebenenfalls einer eindeutigen Beschreibung. Somit ist der Titel immer eindeutig und lässt sich später in der Datenbank auch als Primärschlüssel verwenden (Siehe Abschnitt 3.3.1). Der Titel lässt sich leicht durch den Start- und Endtag `<title>` und `</title>` finden. Die URL des Wikipedia-Artikels lässt sich durch einen immer gleichbleibenden URL-Anfang und dem nachfolgenden Titel bestimmen. Einige Personenartikel enthalten auch eine Kurzbeschreibung. Diese steht in zwei geschweiften Klammern nach einem Pipe-Symbol: `{{Short description|<Kurzbeschreibung>}}`.

Alle weiteren Informationen extrahieren wir aus der Infobox des Artikels. Die englischen Wikipedia-Artikel zu Personen besitzen meistens eine Infobox, in der die wichtigsten Informationen aufgelistet sind. Dies vereinfacht das Finden von Informationen, da sie schon einer Informationsart zugewiesen sind. Eine Zeile mit einer Information ist zumeist wiefolgt aufgebaut: `|informationsart=Information`. Zwischen dem Pipe-Symbol und dem Gleichheitszeichen können außerdem beliebig viele Leerzeichen liegen. Der Mapper prüft jede Zeile, ob sie diesem Aufbau mit einer von uns vordefinierten Informationsart entspricht. Wenn dies der Fall ist, extrahieren wir den Text nach dem Gleichheitszeichen. Dieser Text muss nun noch geparst werden, da er noch viele Sonderzeichen und irrelevante Informationen enthält.

In den Wikipedia-Artikeln gibt es gewisse Elemente, wie z.B. `plainlist` oder `flatlist`, bei denen die eigentlichen Informationen erst in den nachfolgenden Zeilen steht. Wir prüfen also, ob im extrahierten Text ein solches Element vorkommt und verwerfen die Information, wenn dies der Fall ist. Andernfalls entfernen wir XML-Code und einige Sonderzeichen aus dem Text, um ihn von störenden und unwichtigen Inhalten zu säubern. Eckige Klammern beschreiben in den Wikipedia-Artikeln eine Verlinkung. Manchmal steht innerhalb der Verlinkung ein Pipe-Symbol. Der Inhalt vor dem Pipe-Symbol ist der Wikipedia-Artikel auf den verlinkt wird, der Inhalt nach dem Pipe-Symbol ist der Text der angezeigt wird. Wir interessieren uns in diesem Fall für den Inhalt nach dem Pipe-Symbol, da wir die Informationen als einen sinnvollen Satz extrahieren wollen. Mithilfe eines Regex-Ausdrucks findet und entfernt der Mapper die Klammern und alles vor dem Pipe-Symbol. Anschließend wird der Text abhängig von der Informationsart weiterverarbeitet.

In der Infobox steht nur der Name und die Dateiendung des Bildes. Wir erstellen aus dem Bildnamen die Bild-URL, indem wir einen einheitlichen URL-Anfang mit dem Bildnamen verbinden. Leider sind einige Bilder nicht nach diesem Format hochgeladen worden und besitzen einen Link mit willkürlichen Zeichen. Den Link zu diesen Bildern können wir leider nicht herausfinden, da sie keiner einheitlichen Syntax folgen. Da uns nur der Bildname gegeben ist, wissen wir nicht, ob der von uns zusammengesetzte Link auch so existiert. Später in der `main.js` des Backend-Servers fügen wir aus diesem Grund einen `EventListener` hinzu, der nicht-existierende Bilder mit einem Platzhalterbild ersetzt.

Die Daten, wie z.B. das Geburts- bzw. Todesdatum, haben in den Wikipedia-Artikeln keine einheitliche Syntax. Beispielsweise treten Monate manchmal als Zahl und manchmal ausgeschrieben auf. Deswegen enthält der Mapper eine Methode zum Formatieren von Daten in das einheitliche Format `BC/AD-Jahr-Monat-Tag`. Hierfür war es notwendig alle möglichen Datenformate zu prüfen, die das gegebene Datum haben kann.

Aus allen Informationen werden zudem noch weiterer Code und Klammern entfernt bis die Information fertig geparst ist und zu der Informationsliste hinzugefügt werden kann. Die Ergebnismenge des `PersonData` Jobs ist eine Sammlung von Informationen von allen Personen auf Wikipedia. Diese fügen wir zusammen als eine einfache CSV-Datei mit einer speziellen Formatierung. Kommata sind in diesem Fall als Delimiter nicht geeignet, da die Informationen selbst auch Kommata enthalten können. Die CSV-Datei lässt sich durch ihren Aufbau einfach in die Datenbank laden.

TitleExtractor

Der `TitleExtractor` ist ein Hilfsjob für den `Relationships` Job. Er wird vor dem `Relationships` Job ausgeführt und gibt als Ausgabe alle Titel der Wikipedia-Artikel aus. Als Eingabe bekommt er die Ausgabe des `PersonArticleExtractors`, sodass die Liste an Titeln eine Liste aller Personennamen entspricht.

Wie im `PersonData` Job, werden auch hier die Sandbox-Seiten übersprungen, indem geprüft wird, ob der Titel "`user:`" enthält. Die Ausgabe des Jobs fügen wir zu einer TXT-Datei zusammen, die später von dem `Relationships` Job genutzt wird.

Relationships

Der Mapper des `Relationships` Jobs sucht in einem Personenartikel nach Verlinkungen zu anderen Personenartikeln und gibt diese Person zusammen mit dem ersten Satz, in dem sie erwähnt wurde, aus. Der Aufbau der Ausgabe ist ähnlich zu der des `PersonData` Jobs:
`PERSON1>>>>PERSON2>>>>SENTENCE.`

Wie bereits erklärt, werden Verlinkungen durch eckige Klammern beschrieben, die ein Pipe-Symbol enthalten können. Während uns im `PersonData` Job der Inhalt nach dem Pipe-Symbol interessiert hat, interessieren wir uns nun für den Inhalt vor dem Pipe-Symbol, da dies der verlinkte Titel des Wikipedia-Artikels ist.

Um herauszufinden, ob es sich bei dem verlinkten Artikel um einen Personenartikel handelt, nutzt der `Relationships` Job die ausgegebene Textdatei des `TitleExtractor` Jobs. Der Mapper holt sich die Textdatei mit den Titeln aus dem Cache und liest diese Zeile für Zeile ein. Dabei fügt

er die Titel einem HashSet hinzu. Um zu prüfen, ob die Verlinkung auf einen Personenartikel zeigt, muss nun nur noch geschaut werden, ob sich dieser Titel in dem HashSet befindet. Wenn dies der Fall ist, wird der Titel des angeschauten Personenartikels, der Titel des verlinkten Personenartikels und der Satz, in dem die Verlinkung erwähnt wurde, als Output ausgegeben. Den Anfang und das Ende eines Satzes haben wir durch einen Regex-Ausdruck beschrieben, welcher definiert, dass nach einem Punkt ein Leerzeichen und ein Großbuchstabe folgen muss. Dies ist eine zuverlässige Methode, allerdings werden Sätze mit Abkürzungen, wie z.B. “Samuel L. Jackson” falsch interpretiert.

Die Ausgabe des Jobs wird ebenfalls zu einer CSV-Datei zusammengefügt, um die Daten anschließend einfach in die Datenbank zu laden.

3.3 **Backend** (Christian)

3.3.1 Aufbau MySQL-Datenbank

Die MySQL-Datenbank wurde vom base.camp bereitgestellt und konnte nach Erhalt der Zugangsdaten direkt genutzt werden. Die Einrichtung und das Hosten eines eigenen Datenbanksystems ist damit entfallen. Zur weiteren Nutzung unserer Daten und dem späteren Abruf dieser, haben wir zwei Datenbank-Tabellen angelegt. Jede Datenbank-Tabelle wurde aus einer CSV-Datei angelegt, welche die jeweils benötigten Inhalte bereitgestellt hat (Siehe Abschnitt 3.2.2).

Field	Type	Collation	Null	Key	Default	Extra	Privileges	Comment
TITLE	varchar(255)	utf8mb4_unicode_ci	NO	PRI	NULL		select,insert,update,references	
LINK	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
SHORT_DESCRIPTION	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
IMAGE	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
NAME	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
BIRTH_NAME	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
BIRTH_DATE	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
BIRTH_PLACE	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
DEATH_DATE	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
DEATH_PLACE	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
DEATH_CAUSE	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
NATIONALITY	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
EDUCATION	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
KNOWN_FOR	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
OCCUPATION	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
ORDERS	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
OFFICE	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
TERM_START	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
TERM_END	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	
PARTY	text	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	

Abbildung 3.3: Spalten der Tabelle “PersonData”

Die erste Datenbank-Tabelle mit dem Namen `PersonData` enthält alle Personen über die es eine englischsprachige Seite bei Wikipedia gibt. Diese Datenbank stellt die Daten für unseren Zeitstrahl zur Verfügung. In ihr sind, in 20 Spalten, neben dem eindeutigen Titel der Seite, dem Link zur Wikipedia-Seite oder dem Namen der Person, auch alle weiteren wichtigen Informationen zur Person tabellarisch gelistet (siehe Abbildung 3.3), sofern diese Informationen in dem Artikel hinterlegt sind. Da der Titel einer Seite immer eindeutig ist und kurz genug um diesen als Typ `varchar(255)` anzugeben, wurde dieser als Primärschlüssel festgelegt und auch zur Indexierung genutzt. Die anderen Felder konnten nicht zur Indexierung herangezogen werden. Dies begründet sich vor allem aus der Tatsache das sich die Feldlängen, insbesondere der `SHORT_DESCRIPTION` (Kurzbeschreibung der Person), von Person zu Person stark unterscheiden

und des Öfteren die Feldlänge `varchar(255)` nicht ausreichend war, um alle Informationen zu speichern. Felder des Typs `Text` können allerdings nicht als Primärschlüssel festgelegt werden und auch nicht für eine Indexierung genutzt werden. Hier musste eine Abwägung stattfinden, ob wir die Suche in der Datenbank mittels Indexierung weiter beschleunigen wollen oder die Vollständigkeit der Informationsfelder sicherstellen. Die Entscheidung viel hierbei zugunsten der Vollständigkeit aller Informationsfelder und folglich wurden für alle weiteren Felder, ausgenommen `TITLE`, der Typ `Text` gewählt.

In unserer Datenbank sind ca. 1,5 Millionen Personen enthalten. Trotz der von uns gewählten Indexierung, erfolgt eine Suche nach einer einzelnen Person im Regelfall in weit unter einer Sekunde. Natürlich müssen für die spätere Nutzbarkeit nicht nur Übertragungs-, sondern auch Verarbeitungszeit hinzugerechnet werden. Trotzdem war die Suche für uns mit dieser Indexierung hinreichend schnell.

Field	Type	Collation	Null	Key	Default	Extra	Privileges	Comment
PERSON1	varchar(255)	utf8mb4_unicode_ci	NO	PRI	NULL		select,insert,update,references	
PERSON2	varchar(255)	utf8mb4_unicode_ci	NO	PRI	NULL		select,insert,update,references	
SENTENCE	mediumtext	utf8mb4_unicode_ci	YES		NULL		select,insert,update,references	

Abbildung 3.4: Spalten der Tabelle “Relationships”

Die zweite Datenbank-Tabelle `Relationships` stellt die Daten für den Netzwerkgraphen bereit. Diese Datenbank unterteilt sich in drei Abschnitte `PERSON1`, `PERSON2` und `SENTENCE`. Hier ist für jede Person (`PERSON1`) hinterlegt, welche anderen Personen (`PERSON2`) in ihrem Artikel verlinkt sind. Eine Beziehung bedeutet dabei nicht immer, dass die Personen sich auch persönlich kennen, sondern kann auch für eine Erwähnung oder einen Vergleich stehen. Um eine Aussage über die Art der Beziehung zu treffen, wird auch gleichzeitig der Satz hinterlegt in dem die verlinkte Person erwähnt wird (`SENTENCE`). So ist z.B. in dem Artikel von Abraham Lincoln (`PERSON1`) Ann Rutledge (`PERSON2`) verlinkt und die Art der Beziehung wird durch den Satz: „Lincoln’s first romantic interest was Ann Rutledge, whom he met when he moved to New Salem.“ beschrieben und in `SENTENCE` hinterlegt.

PERSON1	PERSON2	SENTENCE
Abraham Lincoln	Ann Rutledge	Lincoln's first romantic interest was Ann Rutledge, whom he met when he moved to New Salem.

Abbildung 3.5: Beispiel eines Eintrags in der Tabelle “Relationships”

Da der Personenname die varchar(255) nicht überschreitet, sind `PERSON1` und `PERSON2` beide vom Typ varchar(255) und auch gemeinsam Primärschlüssel und damit auch direkt indexiert. Da die Beziehungen der Personen in `SENTENCE` den Typ varchar(255) teilweise überschreitet, wurde `mediumtext` gewählt. Eine extra Indexierung war hier aufgrund des Typs nicht möglich bzw. notwendig, da die Beziehung sich nicht zur Indexierung eignet.

Anders als `PersonData` mit 1,5 Millionen Personen, enthält die Datenbank `Relationships` über 3,5 Millionen Beziehungen. Durch die gewählte Indexierung ist die Suche bzw. Ausgabe in dieser Datenbank sehr schnell und gut für das Projekt geeignet.

3.3.2 MySQL-Schnittstelle

Die Klasse `MySQLconnect` ist die Schnittstelle in unserem Programm, das die Authentizierung und den Abruf der Daten vom MySQL-Server sicherstellt. Um die Klasse auch für andere Nutzer direkt nutzbar zu machen, haben wir die Einloggdaten ausgelagert und mit einem entsprechendem Hinweis versehen, sollte versucht werden das Projekt ohne Zugangsdaten für einen MySQL-Server zu starten. Die entsprechenden Einloggdaten müssen im Ordner `resources` als `credentials.txt` hinterlegt werden.

In der Klasse sind neben dem Konstruktor, der den Verbindungsauf- und abbau sicherstellt, auch die drei Methoden `getPersonData()`, `getRelatedPersonData` und `getRelationships` definiert, die entsprechend die Daten für den Zeitstrahl und den Netzwerkgraphen abrufen. Der Abruf der Daten erfolgt allerdings nicht direkt aus der Klasse `MySQLconnect`, sondern indem die Controller-Klassen `PeronController` und `RelatedPersonController` jeweils immer ein Objekt der `MySQLconnect`-Klasse erzeugen und für die jeweilige Datenabfrage nutzen. Diese Lösung wurde gewählt, da die ankommenden Daten vom Datentyp `ResultSet` sind. Diese werden im `ResultSetConverter` in das für beide Einsatzzwecke (Timeline, Netzwerkgraph) benötigte JSON-Format gebracht. Die passende Strukturierung der Daten wird in den Controller-Klassen sichergestellt. Eine fehlgeschlagene Verbindung zur Datenbank wird durch eine Fehlermeldung abgefangen und dem Nutzer angezeigt.

3.4 **Frontend** (Henning)

Das Frontend unserer Anwendung besteht aus einer Webseite, welche dem Nutzer die Visualisierungen der Daten zur Verfügung stellt. Die Webseite insgesamt basiert auf dem Spring-Framework und ist eine Spring-Boot-Anwendung. Der Inhalt und die Logik der Webseite ist in HTML-, JavaScript-, CSS- und Java-Dateien definiert. Im Folgenden sollen nun die verschiedenen Quellcode-Dateien und deren Zusammenspiel erläutert werden.

Die Datei `index.html` definiert den grundlegenden Aufbau der Webseite und basiert dabei auf dem “dashboard”-Beispielprojekt aus Bootstrap 5.0.0. Im `head` werden die Metadaten definiert, sowie Links zu den genutzten CSS- und JavaScript-Librarys angegeben. Im `body` wird dann für kleine Bildschirme eine aus- und einklappbare Seiten-Navigation erstellt. Anschließend wird der eigentliche Inhalt, also die Input-Felder, Buttons, die Buchstaben-Navigation, ein Container für die Timeline-Darstellung, sowie rechts davon die Info-Box erstellt (der HTML-Code-Reihenfolge nach). Weiterhin wird unter dem Timeline-Container der Second-Layer-Button, sowie der zugehörige Container für den Netzwerk-Graphen definiert. Die Datei `about.html` verwendet ebenfalls die Bootstrap-Librarys, um ein einheitliches Design zu gewährleisten. Neben der Seiten-Navigation enthält diese Datei einen Text mit Hintergrund- und Kontaktinformationen zum Projekt.

Die JavaScript-Datei `main.js` ist für die Erstellung der jeweiligen Visualisierungen in den genannten Containern zuständig. Die Funktion `buildTimeline()` erstellt die Timeline-Visualisierung im entsprechenden HTML-Container und die Funktion `buildGraph()` erstellt die Netzwerk-Visualisierung im anderen entsprechenden HTML-Container.
Dabei wird `buildTimeline()` bereits beim initialen Laden der Webseite und beim Klick auf „Search!“ ausgeführt. Die Funktion `buildGraph()` wird bei einem Klick auf eine Person in der Timeline ausgeführt (siehe `main.js`, Zeile 148).

Der Code beider Funktionen, sowie einige der Hilfsfunktionen, wie beispielsweise `zoomIn()`, basieren auf Code-Beispielen aus der AnyChart-Library. Diese haben wir entsprechend unseren Anforderungen im Code konfiguriert. So wird in der `buildTimeline()`-Funktion zunächst die lokale Variable `let url` definiert, welche über das entsprechende Mapping (bspw. `relatedPersons`) auf die entsprechende Controller-Klasse (Java) zugreift. Dabei werden die Filter-Argumente, wie bspw. ein Geburtsdatum oder der Anfangsbuchstabe für einen Personennamen übergeben. Die entsprechende Controller-Klasse gibt die gefilterten Daten zurück. Diese werden dann mit der Funktion `loadJsonFile` weiterverarbeitet (siehe `main.js`, Zeile 34). In dieser Funktion werden sämtliche Konfigurationen an der Visualisierung vorgenommen. So wird insbesondere das Layout und das Design der Timeline angepasst. Besonders wichtig ist das in Zeile 46 vorgenommene Mapping, da die anychart.charts.Gantt-Library die Variablen `name`, `actualStart` und `actualEnd` erwartet. Nach den Funktionen `loadJsonFile` und `buildTimeline()` wird die Funktion `updatePersonalInformationBox(e)` so definiert, dass in der Info-Box, rechts von der Timeline, Informationen zu der in der Timeline angeklickten Person angezeigt werden. Des Weiteren haben wir eine Funktion `dateFormatter` definiert, die die von der Datenbank erhaltenen Geburts- und Todesdaten in das richtige Format bringt. Die Funktion `buildGraph()` ist analog zur erläuterten `buildTimeline()`-Funktion aufgebaut.

Wie bereits beschrieben, werden aus dem JavaScript-Code die Controller-Klassen (Java) aufgerufen. Wir unterscheiden zwischen vier verschiedenen Controller-Klassen: “PageController”, “GraphController”, “PersonController” und “RelatedPersonController”. Nur die drei zuletzt genannten Klassen werden aus dem JavaScript (`main.js`) aufgerufen. Wenn die Funktion `buildTimeline()` durch einen Klick auf den “Search!”-Button aufgerufen wird, fordert sie die Daten von PersonController an. Wenn `buildTimeline()` durch einen Klick auf eine Zeile in der Timeline aufgerufen wird, wird der RelatedPersonController für die benötigten Daten angesprochen. Denn in diesem Fall soll eine Timeline mit Personen, die mit der angeklickten Person in Beziehung stehen, erstellt werden. Wird die Funktion `buildGraph()` ausgeführt, benötigt diese keine Personendaten, sondern Beziehungsdaten. Diese werden vom GraphController zurückgegeben.

In der PersonController- und RelatedPersonController-Klasse wird lediglich ein Objekt der Klasse `MySQLconnect` erstellt, an welchem letztlich die entsprechende Methode zur SQL-Datenabfrage (bspw. `getPersonData`) aufgerufen wird. Die Graph-Controller-Klasse enthält darüber hinaus noch weitere Methoden, welche die aus der `MySQLconnect`-Klasse erhaltenen Daten in die von der AnyChart-Library erwartete JSON-Struktur bringen. Bei ausgewählter “Show SecondLayer”-Option werden die Daten zusätzlich um Daten der zweiten Beziehungsebene ergänzt. Ausganspunkt ist dabei die Methode `converter`. In dieser Methode werden die Daten der ersten Beziehungsebene zunächst in das richtige Format gebracht, indem das JSON-Objekt `obj` erstellt wird. Soll lediglich die erste Beziehungsebene im Netzwerkgraph dargestellt werden (siehe Abbildung 3.6), wird dieses JSON-Objekt aus GraphController an `main.js` zurückgegeben. Soll zusätzlich die zweite Beziehungsebene dargestellt werden (siehe Abbildung 3.7), wird zuvor noch die Methode `getSecondLayer` aufgerufen. In dieser Methode werden die zusätzlichen Beziehungsdaten für die Personen, die mit der ausgewählten Person in Beziehung stehen, angefragt und schließlich im JSON-Objekt `moddedjson` gesammelt. Anschließend wird dieses JSON-Objekt in der Methode `MergeLayer0AndLayer1` mit dem JSON-Objekt für die erste Beziehungsebene, das bereits in `converter` erstellt wurde, zusammengefügt. Dieses finale JSON-Objekt wird dann als String aus dem GraphController an die `main.js` zurückgegeben.

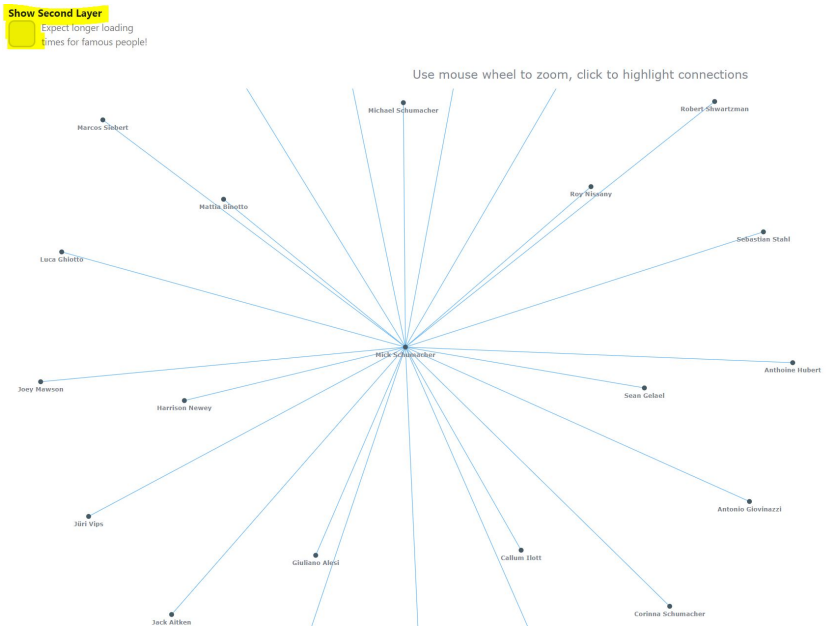

Abbildung 3.6: Daten der ersten Beziehungsebene. Es werden alle Personen dargestellt, die direkt mit der ausgewählten Person (Mick Schumacher) in Beziehung stehen.

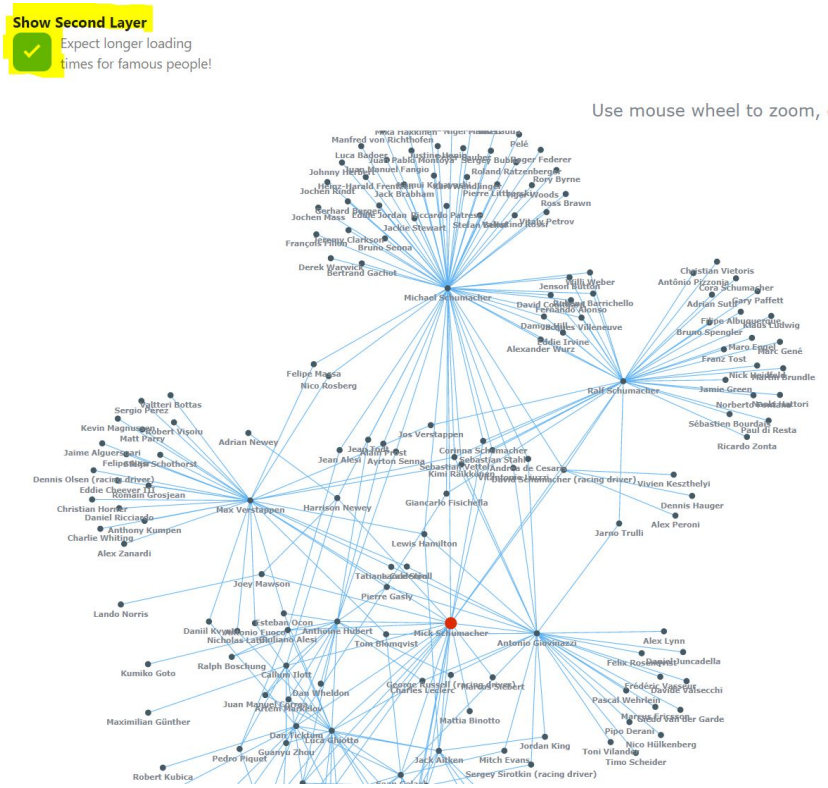

Abbildung 3.7: Daten der ersten und zweiten Beziehungsebene. Es werden wie in Abbildung 3.6 alle direkten Beziehungen zur ausgewählten Person abgebildet. Zusätzlich werden nun die Beziehungen von diesen direkten Beziehungs-Personen abgebildet (Beziehungsdaten der zweiten Ebene).

Die Klasse `MySQLconnect` stellt die Methoden für die Abfrage der Daten von der MySQL-Datenbank zur Verfügung. Der Konstruktor `MySQLconnect()` stellt dabei eine Verbindung zur Datenbank her. Hierzu werden die Zugangsdaten für die Datenbank aus der `credentials.txt` genutzt. Die Methoden `getPersonData`, `getRelatedPersonData` und `getRelationships` erstellen aus den jeweils übergebenen Parametern die entsprechende SQL-Anfrage an eine der beiden MySQL-Tabellen. Sie erhalten von der Datenbank eine Rückgabe vom Datentyp `ResultSet`. Diese muss für eine Weiterverabeitung zunächst in das JSON-Format gebracht werden. Hierzu wird die Klasse `ResultSetConverter` genutzt. Diese basiert auf dem Lösungsvorschlag von Sanil Khurana aus diesem Stackoverflow-Thread.

Die verwendeten CSS-Dateien „bootstrap.min.css“ und „dashobard.css“ wurden von uns weiter angepasst, um beispielsweise die Buttons und Textfelder nach unseren Wünschen darzustellen. Für alle weiteren Implementierungsdetails verweisen wir auf unseren offen zugänglichen Quellcode (Link).

4 Fazit (Christian)

Vorkenntnisse

In unserer Gruppe hatten wir nur Vorkenntnisse in Java. Wir hatten zuvor noch keine Berührungspunkte mit dem Hadoop File System, HTML oder Javascript, was einerseits zu Problemen beim Einstieg, als auch beim späteren Erstellen und Implementieren der Webseite führte. Auf der anderen Seite konnten wir somit viele neue Einblicke gewinnen und Erfahrungen sammeln.

Kommunikation

Durch die coronabedingte Situation studieren wir mittlerweile das dritte Semester in Folge rein digital von zu Hause aus. Entsprechend war es für unsere Gruppe keine große Umstellung, auch das Praktikum digital zu gestalten. Die Treffen und das Organisieren in den Scrum-Meetings waren ein wichtiger Bestandteil während der Präsenzphase des Praktikums. Da wir uns zeitgleich auch über Discord und Whatsapp organisiert hatten, war auch neben den Meetings und insbesondere nach der Präsenzphase eine reibungslose Kommunikation zwischen allen Teammitgliedern möglich.

Zielsetzung

Die grobe Zielsetzung war durch das Praktikum vorgegeben. Nicht nur das Extrahieren aus den Wikipedia Dump-Files sollte funktionieren, sondern die Daten sollten als Webseite aufbereitet und visuell dargestellt werden. Dies konnten wir auch so umsetzen. Als Gruppe haben wir uns dieser Zielsetzung durch das Setzen kleinerer, terminlich festgelegter Zwischenziele immer weiter angenähert. Letzten Endes konnten wir fast alle zusätzlichen Ziele, die wir uns gesetzt haben, erreichen. Nur die Umsetzung des “aufklappbaren” Netzwerkgraphen (siehe Abschnitt 5) haben wir aufgrund der Komplexität der Aufgabe nicht umgesetzt bekommen.

Schwierigkeiten

Durch die fehlenden Vorkenntnisse und die recht komplexen Anforderungen an unsere MapReduce-Jobs, konnten wir, anders als andere Gruppen, nicht einfach auf das bereitgestellte WordCount-Beispiel zurückgreifen und dieses umschreiben. Hier musste von uns ein ganz eigener Ansatz entwickelt werden, um die einzelnen Personen zu extrahieren und die benötigen Daten zu sammeln. Dies führte zu einem hohen Zeitaufwand.

Die von uns gewählte Anychart-Library ist sehr benutzerfreundlich, benötigt jedoch die Daten als JSON-Format. Da wir vorher noch nie mit dem Format gearbeitet haben und insbesondere das Hinzufügen des zweiten Layers zu komplexen JSON-Formatierungen führte, hat dies ebenfalls viel Zeit in Anspruch genommen. Dies führte dazu, dass die Umsetzung des "aufklappbaren" Netzwerkgraphen nicht mehr möglich war. Sonstige kleinere Schwierigkeiten konnten wir gut intern und durch Recherche lösen.

Zusammenfassung

Insgesamt war es ein spannendes Projekt, das uns neben einigen Anstrengungen auch viel Spaß bereitet hat. Dies lag unter anderem auch an Eugen Ruppert, dem Leitenden des Praktikums, der uns bei Problemen und Fragen eine große Hilfe war.

Gerade Projekte die solch eine Teamarbeit erfordern sind während coronabedingter Einschränkungen eine gute Möglichkeit in Kontakt mit anderen Studierenden zu kommen und sich auszutauschen. Solch eine praxisbezogene Teamarbeit sollte aus unserer Sicht auch in anderen Bereichen der Lehre gefördert werden.

5 Ausblick / Mögliche Erweiterungen (Henning)

Auch wenn wir bereits sehr zufrieden mit unserer Webseite sind, bieten sich noch viele Möglichkeiten, um die Features unserer Web-Anwendung zu erweitern. So würde es sich beispielsweise anbieten, dass neben dem Netzwerk-Graphen eine weitere Infobox angezeigt wird, die Informationen zu einer im Netzwerk-Graph markierten Person oder Beziehung anzeigt. Die Beschreibung für eine Beziehung existiert bereits in der Datenbasis und müsste somit nur noch auf der Webseite eingebunden werden.

Ein weiteres nutzerfreundliches Feature wäre das "Aufklappen" des Netzwerk-Graphs. Dabei könnte man sich die Beziehungen einer Person anzeigen lassen, indem man auf den Knoten im Netzwerk-Graphen klickt. In den neu verknüpften Knoten, also die Beziehungen des geklickten Knotens, könnte man wiederum per Klick die jeweiligen Beziehungen anzeigen und so weiter. So könnte man sich praktisch durch das gesamte Personennetzwerk aus Wikipedia klicken. In der von uns angedachten Implementation würde dabei für jeden Knoten-Klick eine neue SQL-Anfrage abgesetzt werden und das vorhandene Datenset um die neuen Beziehungsdaten ergänzt werden, um den Graph neu zu zeichnen.

Ein weiteres nützliches Feature im Netzwerk-Graph wäre die Anzeige von Personen-Bildern, statt Knoten-Symbolen. Weiterhin könnte man die für den Nutzer zugänglichen Filteroptionen erweitern. Beispielsweise um Attribute wie Nationalität oder Ausbildung, die bereits in der Datenbasis vorhanden sind. Auch der bereits vorhandene Geburts- und Todesdatum-Filter ließe sich erweitern, indem man nicht nach einem genauen Datum filtert, sondern nach dem Lebenszeitraum von Personen filtert. Hierzu müsste man hauptsächlich den Datentyp für die Geburts- und Todesdaten in der SQL-Datenbank von String auf Integer ändern.

Neben neuen Features sehen wir insbesondere die aktuelle Performance der Webseite als verbesserungswürdig an. Wird der Button "Show Second Layer" bei sehr berühmten Personen

angewählt, entstehen Ladezeiten von bis zu ungefähr 20 Sekunden. Der Ladevorgang bricht dann ab, weil die Anzahl der Kanten im Netzwerk-Graph auf 500 limitiert ist. Aus unserer Sicht, scheint das verwendete anychart.charts.Graph-Framework nicht für solch riesige Netzwerk-Graphen geeignet, da ein Großteil der Ladezeit nur für das Erstellen des Netzwerk-Graphs mit AnyChart benötigt wird. Ein weiteres Problem bei besonders umfangreichen Netzwerk-Graphen ist, dass die Knoten-Beschriftungen sich stark überlappen und der Graph damit unleserlich wird. Wir hatten bereits versucht, die Schriftgröße zu verringern, was allerdings kleinere Graphen unleserlich machte. Man könnte daher über eine von der Knotenanzahl abhängige Design-Konfiguration nachdenken. Wahrscheinlich ließe sich die Ladezeit verringern, indem wir das im Framework verarbeitete Datenset, um doppelte Einträge bereinigen. Auch das Feature des "aufklappbaren" Graphens könnte die Ladezeiten verkürzen, da nicht die gesamte zweite Ebene auf einmal geladen werden müsste. Sollten diese Maßnahmen nicht zu akzeptablen Ergebnissen führen, kann ebenfalls ein alternatives Framework für die Erstellung des Netzwerk-Graphs in Betracht gezogen werden.