Hochschule Aalen

Projektbericht
Studiengang : Informatik

Visualisierung von MongoDB Datenbanken

von

Max Winter

80559

Betreuender Professor: Prof. Dr. Gregor Grambow

Einreichungsdatum : 01. Dezember 2016

Eidesstattliche Erklärung

Hiermit erkläre ich, **Max Winter**, dass ich die vorliegenden Angaben in dieser Arbeit wahrheitsgetreu und selbständig verfasst habe.

Weiterhin versichere ich, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, dass alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Ort, Datum Unterschrift (Student)

Kurzfassung

In diesem Bericht geht es um die Entwicklung eines Visualisierungstools für MongoDB Datenbanken. Ziel dieses Tools ist es, aus den Dokumenten und Collections einer MongoDB Datenbank Schemas zu extrahieren, welche daraufhin anschaulich visualisiert werden. Dieses Tool ist Teil eines größeren Datenbank Toolkits, weshalb großer Wert auf Modulatität und Erweiterbarkeit gelegt wird.

Das MongoDB Visualisierungstool besteht aus einem React Frontend und einem Flask Backend, welche über HTTP miteinander kommunizieren. Die Aufgabe des Backends ist es, sich mit einer MongoDB Datenbank zu verbinden. Diese Datenbank wir bei erfolgreicher Verbindung analysiert und das Ergebnis der Analyse als JSON zurückgegeben. Das Frontend visualisiert diese Daten daraufhin in Form von Tabellen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Abkürzungsverzeichnis

1. Einleitung

1.1. Motivation

Für einen Entwickler, der mit einer Datenbank arbeitet, ist es wichtig zu wissen, wie diese Datenbank aussieht, was für Abhängigkeiten es gibt, und ob die Implementierung auch wirklich der Planung entspricht. Aus diesem Grund werden Visualisierungs- und Analysetools für Datenbanken benötigt. MongoDB hat sich in den letzten Jahren zu einem der wichtigsten Datenbanksysteme entwickelt, da aufgrund immer größerer werdenden Datenmengen die Vorteile von NoSQL-Datenbanken für immer mehr Anwendungen überwiegen. [10] Da MongoDB keine relationale Datenbank ist, können herkömmliche Visualisierungs- und Analysetools, die für SQL entwickelt wurden, nicht verwendet werden. Für MongoDB gibt es zwar Visualisierungstools, wie Beispielsweise MongoDB Charts und MongoDB Compass, die meisten davon visualisieren aber die Daten in der Datenbank, und nicht die Struktur und das Schema der Dokumente in der Datenbank. [1] Die Daten selbst können jedoch sehr vielzahlig sein, was ein vollständiges Überblicken der Datenbank schwierig bis unmöglich macht. Auß diesem Grund soll das Ziel dieser Arbeit sein, solch ein Visualisierungstool zu entwickeln.

1.2. Problemstellung und -abgrenzung

Es soll ein MongoDB Visualisierungstool entwickelt werden, welches eine MongoDB Datenbank analysiert und auswertet. Daraus sollen Schemas extrahiert und visualisiert werden. Das Visualisieren der konkreten Daten in der MongoDB Datenbank ist nicht Teil des Problems.

1.3. Vorangegangene Arbeit

Das MongoDB Visualisierungstool soll in eine vorangegangene Arbeit integriert werden. In dieser vorangegangenen Arbeit wurde ein Entity-Relationship Modellierungstool entwickelt. Einer der Ziele dieses Modellierungstools war es, die Anwendung möglichst modular zu gestalten, damit das Modellierungstool langfristig zu

einem umfassenden Datenbank-Toolkit erweitert werden kann. Ein Hauptfokus des MongoDB Visualisierungstools liegt deshalb darauf, diese Modularität beizubehalten und gegebenenfalls weiter zu verbessern.

1.4. Ziel der Arbeit

Ziel dieses Projekts ist es, ein Visualisierungstool für MongoDB Datenbanken zu entwickeln, welches die Dokumente einer Datenbank analysiert und auswertet. Daraus sollen Schemas abgeleitet werden, welche anschließend visualisiert werden. Dies erleichtert es den Entwicklern der Datenbanken, die Strukturen und Abhängigkeiten in ihren Datenbanken zu verstehen und zu verbessern. Dafür soll eine Backendanwendung zur Verbindung und Analyse von MongoDb Datenbanken entwickelt werden, sowie ein Webfrontend für die Visualisierung der analysierten Daten. Das Visualisierungstool soll dabei in die vorangegangene Arbeit integriert werden und die Gesamtlösung dabei modular halten.

Um Datenbanken analysieren zu können, muss es möglich sein, sich mit diesen zu verbinden. Dies erfordert einerseits eine Nutzeroberfläche, über die der Nutzer die Verbindungsdaten eingeben kann. Andererseits erfordert dies die Verbindung mit der Datenbank selbst über eine geeignete MongoDB Schnittstelle im Backend. Die Dokumente der Datenbank müssen anschließend in möglichst kurzer Zeit analysiert werden, um aus ihnen Schemas zu extrahieren. Diese Schemas müssen dann in einem Frontend übersichtlich und visuell ansprechend angezeigt werden. Dafür werden hauptsächlich 2 Ansichten benötigt: Einerseits wird eine Übersicht über alle Collections benötigt, in welcher für jede Collection das meistverwendete Schema angezeigt werden soll. Andererseits wird für jede Collection eine Detailansicht benötigt, welche alle Schema-Variationen in einer Collection sowie weitere Details und Daten anzeigt.

2. Grundlagen

2.1. HTTP

Hypertext Transfer Protocol (HTTP) ist ein Protokoll, welches benutzt wird, um über das Internet zu kommunizieren. Hauptsächlich wird HTTP für die Kommunikation zwischen einem Webbrowser und einem Webserver genutzt. In HTTP wird mittels Nachrichten kommuniziert. Es wird erst ein Request vom Client abgesetzt, der daraufhin von dem Server mit einer Response beantwortet wird. Nachrichten bestehen aus einem Header und einem Body. Der Body enthält den Inhalt der Nachricht. Der Header enthält Meta-Daten über einen Request, wie beispielsweise die angefragten Ressourcen und Datentypen des Body.

2.2. REST API

Ein Application Programming Interface (API) ist eine Schnittstelle, über die von außen mit einem Programm interagiert werden kann. Representational State Transfer (REST) ist ein Protokoll, welches die Kommunikation über HTTP spezifiziert. REST sieht ein Backend vor, welches Resourcen beinhaltet. Ein Client kann über die gängigen HTTP Operationen mit diesen Resourcen interagieren. Jede Resource hat eine global eindeutige ID und wird meist durch JavaScript Object Notation (JSON) oder Extensible Markup Language (XML) repräsentiert. In Java wird ein RESTful Webservice für gewöhnlich mit Jakarta RESTful Web Services (JAX-RS) umgesetzt. [16]

2.3. SQL

Die Structured Query Language (SQL) ist eine Datenbanksprache für relationale Datenbanken. In SQL werden die Daten in Tabellen organisiert, die ein festes Schema definieren. Man kann SQL in 2 Teile aufteilen: Die Data Definition Language (DDL) definiert den Aufbau des Schemas. Mit ihr können Datenbankobjekte erzeugt und gelöscht werden. Zur DDL gehören unter anderem die Befehle CREATE, ALTER, DROP und TRUNCATE. Mit der Data Manipulation Language (DML) können Daten

manipuliert, also eingefügt, geändert, gelesen und gelöscht werden. Die Befehle hierfür lauten SELECT, INSERT, UPDATE, DELETE, MERGE und noch weitere. [15]

2.4. JSON

JSON ist ein Format zum Datenaustausch. JSON basiert auf der Syntax von JavaScript Objekten, ist jedoch unabhängig von der Programmiersprache einsetzbar. Vorteile von JSON sind die einfache Lesbarkeit für Menschen und das einfache parsen und generieren für Maschinen. In JSON gibt es unter anderem folgende Datentypen: [8]

Object ist ein Set aus Key/Value Paaren.

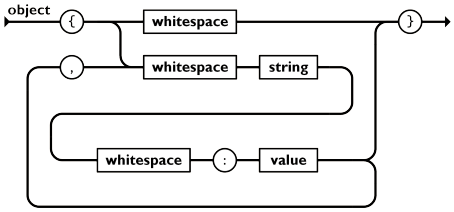

Abbildung 2.1.: JSON Object

Array ist eine geordnete Liste von Values.

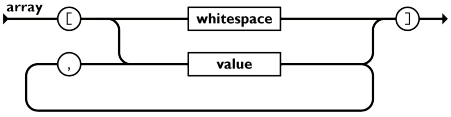

Abbildung 2.2.: JSON Array

Value kann ein String, eine Zahl, ein Boolean, ein Objekt, ein Array oder null sein. Dabei können beliebig viele Values ineinander verschachtelt sein.

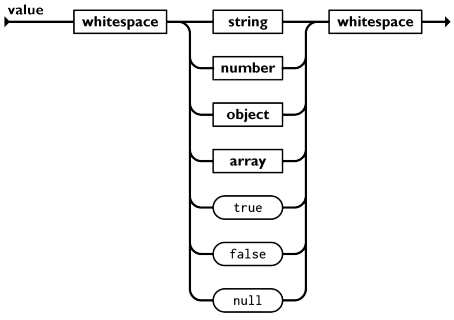

Abbildung 2.3.: JSON Value

String ist eine Sequenz aus Unicode Buchstaben.

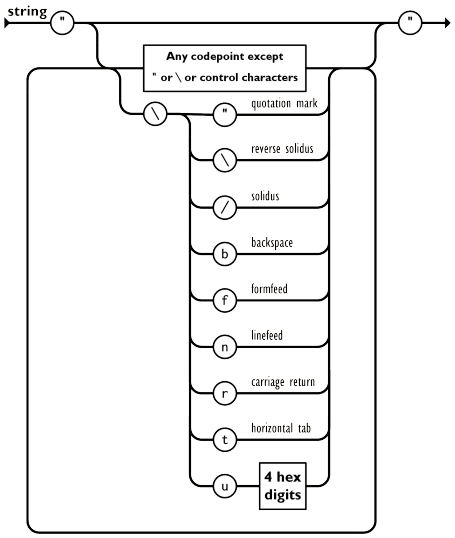

Abbildung 2.4.: JSON String

Number ist eine Zahl. Number kann sowohl eine Gleitkommazahl als auch eine Ganzzahl sein, und kann positiv sowie negativ sein.

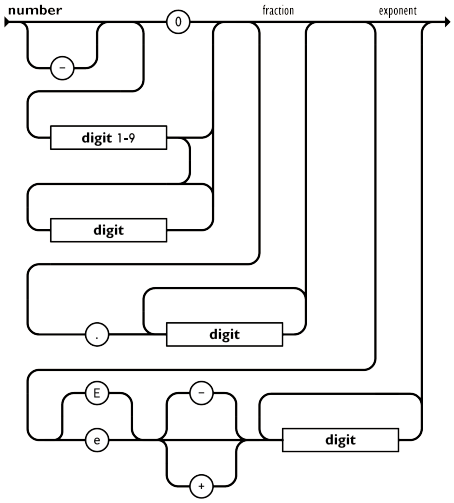

Abbildung 2.5.: JSON Number

2.5. MongoDB

MongoDB ist eine Dokument-orientierte Datenbank. In Dokument-orientierten Datenbanken wird das aus SQL bekannte Konzept von Reihen durch Dokumente ersetzt. Dokumente haben im Gegensatz zu Reihen in Tabellen kein fixes Schema, welches sie erfüllen müssen, sondern sind sehr flexibel. Grundsätzlich bestehen Dokumente aus Key - Value Paaren, welche in einer JSON-Ähnlichen Struktur gespeichert werden. Dokumente können, wie auch in JSON, ineinander verschachtelt sein, was hierarchische Strukturen und dadurch Denormalisierung ermöglicht. Die Dokumente werden in Collections organisiert. Eine Collection entspricht in SQL einer Tabelle ohne das fixe Schema. Diese Collections befinden sich wiederum in Datenbanken. Eine MongoDB Instanz kann mehrere voneinander unabhängige Datenbanken beinhalten. Man kann sich mit der MongoShell mit einer MongoDB Instanz verbinden, um mit der Mongo Query Language oder mit JavaScript die Instanz administrieren und Daten manipulieren zu können. [4]

2.6. Python

Python ist eine objektorientierte high-level Programmiersprache, die besonderen Wert auf Lesbarkeit legt. Variablen in Python werden dynamisch typisiert. Das bedeutet, dass eine Variable in Python keinen festen Typ hat, sondern dieser dynamisch über den ihr zugewiesenen Wert bestimmt wird. Python ist keine compilierte, sondern eine interpretierte Programmiersprache. Diese Eigenschaften machen Python zu der idealen Sprache für das Schreiben von Skripten, sowie für Anwendungen, die schnell und simpel entwickelt werden sollen. PIP ist ein in Python integrierter Paketmanager, der das installieren von Packages erleichtert. Der Python Interpreter ist in C geschrieben, weshalb man die Funktionalität des Interpreters durch C-Programme erweitern kann, um beispielsweise laufzeitkritische Funktionen in C auszuführen. [20]

2.7. Flask

Flask ist ein minimalistisches Open-Source Web Framework für Python. In Flask sind ein paar wenige Kernpakete vorinstalliert, welche für ein minimales Backend benötigt werden. Alles weitere muss der Nutzer selbst über PIP installieren. Durch diesen minimalistischen Ansatz ist Flask sehr flexibel einsetzbar. Flask ist beispielsweise mit SQL oder NoSQL Datenbanken, aber auch ohne Datenbanken einsetzbar. [6]

2.8. JavaScript

JavaScript ist eine High-Level Programmiersprache, die just-in-time kompiliert wird. JavaScript ist besonders bekannt als Skriptsprache für Webseiten, wird aber auch für Backendanwendungen, Automatisierungsaufgaben und vieles mehr verwendet. Die Pakete in JavaScript werden mittels dem Paketmanager npm verwaltet.

2.9. React

React ist eine JavaScript Bibliothek zum Bauen von Benutzeroberflächen. React ist Komponentenbasiert. Das bedeutet, das React aus einzelnen Komponenten bestehen, die ihren eigenen State haben, und die zusammengesetzt ein komplexes User Interface (UI) bilden. [3]

2.9.1. Redux Store

Redux ist ein State Container für JavaScript Apps, der es ermöglicht, States nicht mehr in den einzelnen React Komponenten, sondern in einem zentralen Container zu speichern. Dadurch können States wiederhergestellt werden, nachdem eine Komponente geschlossen und wieder geöffnet wurde. Außerdem erleichtert Redux es, State Änderungen in Komponenten nachzuvollziehen. [5]

2.9.2. Axios

Axios ist ein HTTP-Client für JavaScript. Mithilfe von Axios Können HTTP-Requests versendet und die Responses von diesen verarbeitet werden.

2.9.3. Docker Container

Ein Container ist eine virtuelle Umgebung, die, ähnlich wie eine virtuelle Maschine, eine Anwendung in eine isolierte Umgebung verpackt. Jedoch wird im Gegensatz zu virtuellen Maschinen nicht das ganze Betriebssystem virtualisiert, sondern nur einzelne Anwendungen. Dadurch ist ein Container deutlich schlanker als eine virtuelle Maschine. Container können dadurch als sogenannte Images auf verschiedenen Systemen ausgeliefert werden, und dabei aufgrund der Isolierung unabhängig vom System zuverlässig funktionieren. Docker ist eine weit verbreitete Open-Source Software, welche das Container Prinzip implementiert. [2]

2.9.4. Nginx

Nginx, ausgesprochen Engine X, ist ein Open-Source Web Server. In Nginx sind einige Features für das Hosten von Webservern integriert, wie beispielsweise Load Balancing, Reverse Proxies, Web Sockets, Mail Server und vieles mehr. Dabei unterstütz Nginx alle gängigen Betriebssysteme, wie Windows, Windows Server, MacOS und Linux. Die Konfiguration in Nginx erfolgt über Konfigurationsdateien. Die Nutzung von Nginx als Webserver nimmt jedes Jahr weiter zu im Vergleich zum Kompetitoren wie Apache. Im Januar 2023 liefen 21,20% der meistgenutzten Seiten über Nginx. [13]

3. Problemanalyse

An das MongoDB Visualisierungstool gibt es eine Reihe von Anforderungen, welche erfüllt werden müssen, damit das Projekt als Erfolg gewertet werden kann. Diese werden im Nachfolgenden nach Frontend und Backend getrennt analysiert.

3.1. Anforderungen an das Frontend

Das Resultat des Projekts soll eine Gesamtlösung sein, die aus dem bestehenden ER Modellierungstool und dem MongoDB Visualisierungstool besteht. Diese Gesamtlösung soll flexibel durch weitere Anwendungen und Funktionalitäten erweitert werden können. Das Frontend des ER-Modellierungstools besteht aus einer React Webapp. Deshalb muss das Frontend der Gesamtlösung ebenfalls in einer gemeinsamen Webapp ausgeliefert werden (**FA1**). Diese Webapp muss in der Paketstruktur sowie in den verwendeten Komponenten modular sein (**FA2**). Ohne diese Modularität wäre die Erweiterbarkeit des Frontends nicht gegeben. Um auf die einzelnen Anwendungen der Gesamtlösung zugreifen zu können, wird ein Startbildschirm benötigt, von welchem aus man die Applikationen starten kann (**FA3**).

Da das MongoDB Visualisierungstool ein relativ kleines Tool ohne großen Funktionsumfang ist, sind die meisten Nutzer nicht bereit, erst ein Handbuch für die Benutzung der Anwendung zu lesen. Aus diesem Grund muss die Anwendeung intuitiv benutzbar sein. Das bedeutet, dass alle Funktionen der Anwendung selbsterklärend, oder in der Anwendung selbst ausreichend beschrieben sein müssen (**FA4**). Dies erfordert auch, dass alle Frontend-Anwendungen der Gesamtlösung sich gleich benutzen lassen und ein einheitliches Design verwenden, da dies sonst den Arbeitsfluss und dadurch die intuitive Bedienung behindert (**FA5**).

MongoDB Dokumente können beliebig groß und beliebig verschachtelt sein. Ein Dokument kann eingebettete Dokumente, sowie Arrays in belibiger Tiefe ineinander geschachtelt haben. Damit die Visualisierung großer Dokumente trotzdem einen Mehrwert hat, müssen diese unabhängig von der Größe übersichtlich und verständlich sein. Aus diesem Grund ist eine übersichtliche Darstellung der Schemata sehr wichtig (**FA6**).

Die Dokumente in einer Collection müssen nicht zwangsläufig das gleiche Schema haben. Diese Schema Abweichungen können Verschiedene Ursachen haben: Das

Schema kann einige Optionale Felder haben, welche in manchen Dokumenten nicht gesetzt sind. Es könnte sich aber auch um verschiedene Versionen eines Schemas handeln, das sich im Laufe der Entwicklung gewandelt hat. Es könnte sich bei den Abweichungen aber auch um Fehler in der Implementierung handeln. Für einen Entwickler kann es deshalb hilfreich sein, einen Überblick über alle Variationen in einer Collection zu haben. Aus diesem Grund soll das MongoDB Visualisierungstool für jede Collection eine Detailansicht haben, die diese Variationen visualisiert. Der Unterschied in diesen Variationen kann vor allem bei großen Dokumenten unter Umständen nicht direkt ersichtlich sein. Aus diesem Grund müssen diese Abweichungen vom Hauptschema deutlich hervorgehoben werden.(**FA7**)

3.2. Anforderungen an das Backend

Das MongoDB Visualisierungstool muss auch sehr große MongoDB Datenbanken analysieren können. Vor allem bei großen Datenbanken ist ein Visualisierungstool besonders hilfreich, da große Datenmengen sehr schwer Überschaubar sind. Die Analyse sollte jedoch nicht länger als ein paar Sekunden dauern, da dies den Arbeitsfluss der Benutzer unterbrechen würde. Ein Entwickler sollte während der Arbeit an der Datenbank das Visualisierungstool schnell starten können, um ihn bei der Analyse der Datenbank zu unterstützen. Deshalb müssen die Datenbanken möglichst performant analysiert werden. Die Laufzeit der Analyse sollte 10 Sekunden nicht überschreiten (**BA1**).

Die Webapp kann potenziell von beliebig vielen Personen gleichzeitig benutzt werden. Dies bedeutet, dass das Backend des MongoDB Visualisierungstools mehrere Anfragen gleichzeitig abarbeiten können muss. Deshalb müssen mehrere MongoDB Datenbanken gleichzeitig verbunden und analysiert werden könnnen (**BA2**).

baum

3.3. Übersicht über die Anforderungen

Frontend:

FA1 Gemeinsame Webapp aller Anwendungen der Gesamtlösung

FA2 Modularität

FA3 Startbildschirm

FA4 Intuitive Benutzbarkeit

FA5 Einheitliches Design und Layout

FA6 Übersichtliche Darstellung der Schemata

FA7 Visualisierung der Schema-Variationen in einer Collection

Backend:

BA1 Analyse der Datenbanken in unter 10 Sekunden

BA2 Verbindung und Analyse mehrerer MongoDB Datenbanken gleichzeitig

4. Lösungskonzept

4.1. Bestehende Visualisierungstools

Es gibt bereits MongoDB Visualisierungtools auf dem Markt, jedoch erfüllt keines davon die zuvor definierten Anforderungen zu genüge: [1]

4.1.1. MongoDB Data Explorer

MongoDB Data Explorer ist ein Tool, welches in MongoDB Atlas integriert ist. MongoDB Atlas ist ein Web-Tool zur Verwaltung von MongoDB Datenbanken. Mittels dem MongoDB Data Explorer kann man die Dokumente, Collections und Indexe einer Datenbank anschauen, sowie die Daten mit CRUD Operationen verwalten. Jedoch bietet der MongoDB Data Explorer keine Möglichkeiten, die Schemas der Dokumente zu analysieren.

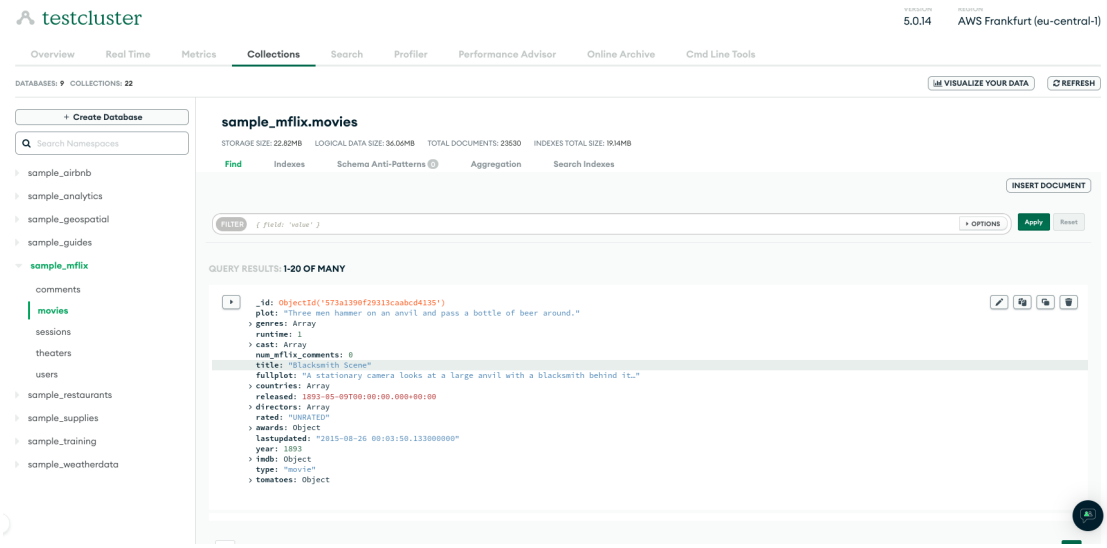

Abbildung 4.1.: MongoDB Data Explorer

4.1.2. MongoDB Compass

MongoDB Compass ist eine Desktop-Anwendung zur Analyse von MongoDB Datenbanken. MongoDB Compass besitzt ein Schema-Visualisierungstool. Dieses Schema-Visualisierungstool zeigt sehr genaue Daten zu jedem Feld der Dokumente einer Collection an. Diese Ansicht ist jedoch nicht besonders übersichtlich, wenn man das gesamte Schema der Dokumente einer Collection analysieren will. Ebenfalls nicht gut ersichtlich in MongoDB Compass ist die Varianz im Schema zwischen den Dokumenten in einer Collection.

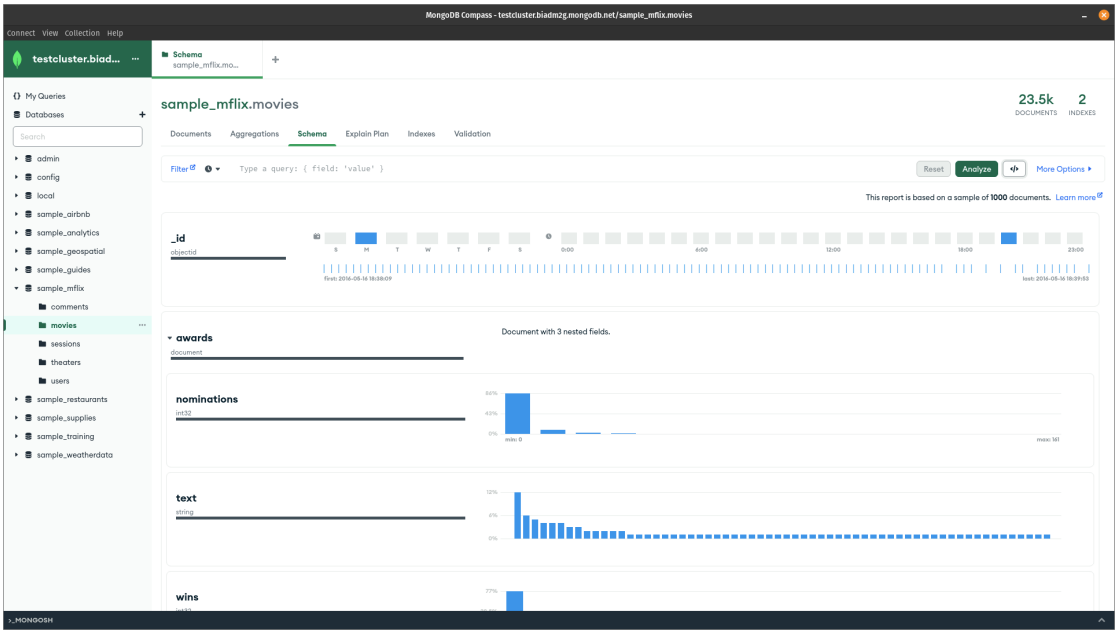

Abbildung 4.2.: MongoDB Compass

4.1.3. MongoDB Charts, Tableau, Qlik und Looker

MongoDB Charts, Tableau, Qlik und Looker sind Tools, die aus MongoDB Daten Graphen generieren und dadurch die Daten in einer MongoDB visualisieren können. Diese Tools konzentrieren sich jedoch alle auf die Visualisierung der Daten, nicht die Analyse und Visualisierung der Schemas der Dokumente.

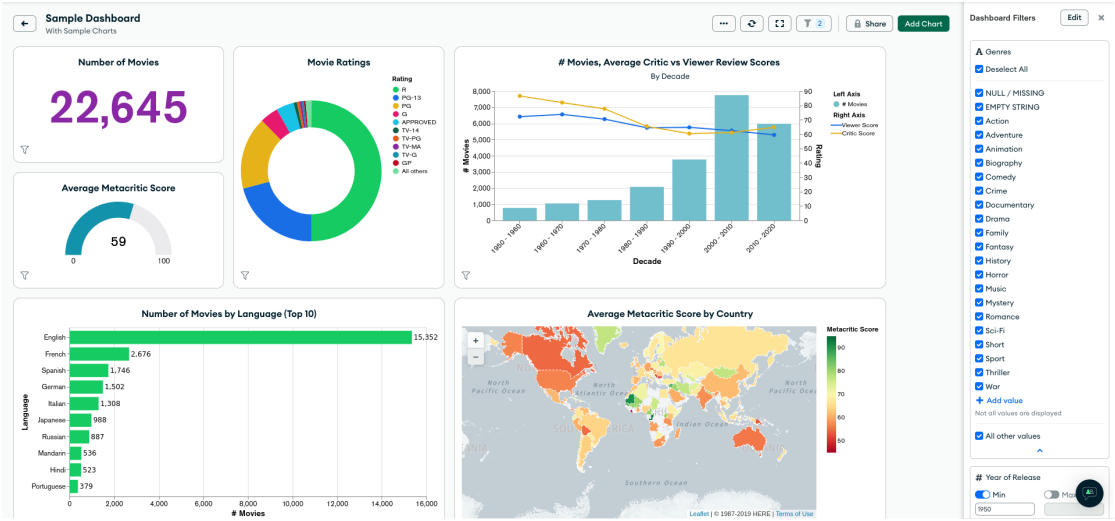

Abbildung 4.3.: MongoDB Charts

4.1.4. Vergleich der bestehenden Tools

In der Nachfolgenden Tabelle wird verglichen, welche Anforderungen erfüllt werden, und welche nicht. Dabei wurden alle Anforderungen ignoriert, welche sich auf die Gesamtlösung des Database Toolkits bezogen, da diese nicht auf spezialisierte Anwendungen anwendbar sind. Ebenso weggelassen wurde die Anforderung **BA2** "Verbindung und Analyse mehrerer MongoDB Datenbanken gleichzeitig. Da alle Anwendungen von beliebig vielen Nutzern gleichzeitig benutzbar sind, ist diese Anforderung überall Erfüllt. Die Bestimmung der Analysedauer für die Anforderung **BA1** wurde exemplarisch mit der größten Datenbank der MongoDB Atlas Beispieldatenbanken (sample_training) durchgeführt.

	MongoDB Data Explorer	MongoDB Compass	MongoDB Charts & Co.
FA1: Webapp	✓	X	✓
FA4: Intuitiv Benutzbar	✓	✓	X
FA6: Übersichtliche Darstellung der Schemata	X	X	X
FA7: Visualisierung der Schema-Variationen	X	X	X
BA1: Analyse der Datenbanken in unter 10 Sekunden	-	?	-

Der MongoDB Data Explorer führt gar keine Analyse-Dauer durch, deshalb kann die Analysedauer für **BA1** nicht bestimmt werden. MongoDB Compass analysiert

immer nur einzelne Collections der Datenbank, und nicht ganze Datenbanken auf einmal, weshalb sich hier die Zeit ebenfalls nicht bestimmen lässt. MongoDB Charts analysiert, wie auch der Data Explorer, keine Schemas, weshalb es hier ebenfalls keine Analysedauer gibt.

Keine der Anwendungen erfüllt die Anforderungen **FA6** und **FA7**. Das Bedeutet, dass keine der Anwendungen die Schemas ausreichend Übersichtlich darstellt, und somit keine der Anwendungen die Problemstellung erfüllt.

4.2. Verwendete Technologien

4.2.1. Backend Technologien

Da die Analyse der MongoDB Dokumente sehr rechenintensiv ist, wird die Analyse in ein Backend ausgelagert. Um den zuvor definierten Anforderungen gerecht zu werden, ist es wichtig, ein geeignetes Backend-Framwork auszuwählen. Das ER-Modellierungstool nutzt Java Spring als Backend-Framework. Da man zum Teil Code von dem bestehenden Backend übernehmen könnte, bietet es sich deshalb an, in diesem Projekt ebenfalls Spring zu verwenden.

Für Spring gibt es eine MongoDB Implementierung namens Spring Data MongoDB. Diese Implementierung ist jedoch dafür ausgelegt, Plain Old Java Object (POJO)s auf Dokumente zu mappen. Im MongoDB Visualisierungstool sollen hingegen MongoDB Dokumente dynamisch eingelesen und analysiert werden. Um **BA2** zu erfüllen, ist es desweiteren nötig, beliebig viele verschiedene Datenbanken gleichzeitig zu verbinden und zu analysieren. Die Verbindung mit MongoDB Datenbanken in Spring Data MongoDB erfolgt jedoch mit fixen Datenbanken, welche in der application.properties Datei definiert werden. [17] Deshalb ist die Spring Data MongoDB Bibliothek für diese Anwendung nicht geeignet. Neben der Spring Data MongoDB Bibliothek gibt es auch noch einen anderen MognoDB Java Client, Java Sync. Dieser funktioniert jedoch nicht zusammen mit dem Spring Framework. Aus diesem Grund kann Spring sowie andere Java Backend Frameworks nicht genutzt werden.

Als alternatives Backend Framework mit REST Api bietet sich Flask an. Ein großer Vorteil von Flask ist, dass Flask sehr minimal ist und nur mit dem minimum an benötigten Bibliotheken vorkonfiguriert ist. Spring ist im Gegensatz dazu ein sehr mächtiges Framework mit vielen Features, von denen in diesem Projekt aber nur sehr wenige gebraucht werden. Ein weiterer Vorteil von Flask sowie von Python ist die Schlankheit des Codes. In Python lässt sich meist die gleiche Funktionalität in weniger Code schreiben als in Java. Dazu kommt, dass in Flask sehr viel weniger Boilerplate Code benötigt wird als in Spring. Ein minimaler Endpunkt in Flask lässt sich bereits mit 2 Zeilen Code umsetzen. Zudem ist die dynamische Typisierung

in Python beim Auswerten der MongoDB Dokumente von Vorteil, da man im Voraus nicht weiß, welche Datentypen die Werte in den Dokumenten haben, und die dynamische Typisierung deshalb das Handling dieser Werte vereinfacht. [9]

Jedoch hat Flask nicht nur Vorteile gegenüber Spring: Flask ist grundsätzlich deutlich unperformanter als Spring. Dies liegt unter anderem daran, dass Python eine interpretierte Sprache ist, und Java eine kompilierte. [18] Dies widerspricht zunächst der Anforderung **BA1**. Die Performance-Probleme lassen sich aber durch Multiprocessing ausgleichen. Multiprocessing bedeutet, dass bestimmte Teile der Berechnung auf mehrere Threads im Prozessor aufgeteilt werden und dadurch parallell ausgeführt werden. Python bietet eine simpel zu implementierende Lösung für Multiprocessing an, welche man bei der Analyse der Dokumente der MongoDB Datenbanken gut einsetzen kann. Beispielsweise kann die Analyse jeder Collection von einem extra Thread ausgeführt werden. Dadurch lässt sich die Anforderung **BA1** mit Flask erfüllen.

In Python gibt es die Bibliothek PyMongo, welche alle der genannten Nachteile von Spring Data MongoDB ausbessert: Mit PyMongo kann man direkt im Code beliebig viele MongoDB Datenbanken parallel dynamisch einbinden. Mittels Objektorientierung lässt sich dadurch die parallele Analyse mehrerer Datenbanken sinnvoll umsetzen. Zudem ist PyMongo nicht für das Mappen von Dokumenten auf Objekte gedacht. Stattdessen kann man Dokumente als Python Dictionary auslesen. Dies erleichtert die Analyse der Dokumente und hat darüber hinaus den Vorteil, dass man Dictionaries in Python in JSON umwandeln kann, was das Bauen der HTTP Response vereinfacht. [14]

4.2.2. Frontend Technologien

Web Apps haben gegenüber Desktop Apps einige Vorteile: Web Apps müssen nicht installiert werden, sie müssen nicht für mehrere Betriebssysteme entwickelt werden und der Auslieferungs- Update- und Administrierungsprozess ist deutlich vereinfacht. Jedoch haben Web Apps oftmals nicht die Interaktionsmöglichkeiten von Desktop Apps, da sie innerhalb eines Browsers laufen. Dies ist in dieser Anwendung jedoch kein großer Nachteil, da die Hauptaufgabe der Anwendung die Visualisierung von Daten ist, und dies nicht viele Interaktionsmöglichkeiten erfordert. [21] Deshalb wird das Frontend dieser Anwendung als Webapp entwickelt.

Das ER Modellierungstool benutzt das Frontend Framework React. Da das ER Modellierungstool und das MongoDB Visualisierungstool Teil eines Datenbank Toolkits werden sollen, muss das MongoDB Visualisierungstool Frontend ebenfalls in React geschrieben werden, damit Anforderung **FA1** erfüllt werden kann. Da React dank React Elements und React Components in seiner Grundstruktur sehr modular ist, eignet sich React sehr gut, um Anforderung **FA2** zu erfüllen. [3] Die

Tools lassen sich mittels Ordner strukturell voneinander trennen, und trotzdem können die Tools sich Komponenten teilen und diese wiederverwenden.

Dank der Komponentenbibliothek Material UI kann man in React vordefinierte Elemente benutzen, was oftmals das Definieren der Komponenten von Hand erspart. Dadurch spart man sich einerseits Programmieraufwand, andererseits verringert dies aber auch die Code-Komplexität und verbessert somit die Lesbarkeit des Codes. Zudem erleichtert Material UI das Umsetzen eines einheitlichen Designs, da man mithilfe von Material UI global verwendbare Themes erstellen kann. Dies ermöglicht das Erfüllen der Anforderung **FA5**. [11]

4.3. REST-Schnittstelle

Das Backend hat eine einzige Funktion: Das Verbinden und Analysieren von MongoDB Datenbanken. Aus diesem Grund stellt Das Backend nur einen einzigen Endpunkt bereit: /*connect* erwartet im Body des Requests folgende Daten im JSON-Format:

- connection_string: Der Connection String zur Verbindung mit der MongoDB Instanz
- database: Der Name der Datenbank, die analysiert werden soll
- analyse_ref: True, wenn die Referenzen der Datenbank analysiert werden sollen
- sort_method: Bestimmt, wie die Dokument-Variationen in einer Collection sortiert werden sollen

Mithilfe dieser Variablen versucht das Backend, sich mit der MongoDB Datenbank zu verbinden. Dafür erstellt es ein Objekt der Klasse DatabaseAnalysis, und ruft die entsprechenden Operationen in diesem Objekt auf. Wenn die Verbindung erfolgreich, wird die Datenbank ausgewertet und das Ergebnis als JSON im Response-Body zurückgegeben. Wenn die Verbindung fehlschlägt, wird der Statuscode 406 zurückgegeben. 406 ist in diesem Fall ein Willkührlich gewählter, für gewöhnlich selten verwendeter Response Code, da HTTP keinen Response Code für das Fehlschlagen von Verbindungen mit Drittanbietern vorgesehen hat.

4.4. Analyse der MongoDB Datenbank

Dokumente in MongoDB sind in einem JSON-Ähnlichen Format geschrieben. Da man in Python JSON zu Dictionaries konvertieren kann, kann man mithilfe von

PyMongo Dokumente als Dictionaries auslesen. Dictionaries beinhalten eine Reihe an Key-Value Paaren. Jedes dieser Key-Value Paare wird ausgewertet, und das Ergebnis der Auswertund in dem Objekt Value gespeichert. Der Key entspricht dem Name des Feldes und wird direkt im Value-Objekt abgespeichert. Aus dem Value des Key-Value Paares wird der Datentyp auslesen. Wenn der Datentyp des Values Embedded Document oder Array entspricht, werden rekursiv die Key-Value Paare aus dem Value ausgelesen. Diese rekursiven Verschachtelungen werden ebenfalls in dafür vorgesehenen Variablen im Value-Objekt gespeichert Falls es sich um eine ObjectId handelt, und nicht den Key _id besitzt (Also kein Primary Key ist), dann wird davon ausgegangen, dass es sich bei dem Value um eine Referenz handelt. Je nachdem, ob der Boolean analyse_ref true ist oder nicht, wird daraufhin versucht, die referenzierte Collection zu finden.

Alle Values eines Dokuments werden in einem ProcessedDocument Objekt gespeichert, welches das Schema des Dokuments repräsentiert. Die Klasse ProcessedCollection speichert wiederum eine Liste an ProcessedDocuments. Diese ProcessedDocuments sind einmalig. Wenn ProcessedDocuments mehrfach vorkommen, dann wird in das bestehende ProcessedDocument die Anzahl der Dokumente mit diesem Schema geschrieben.

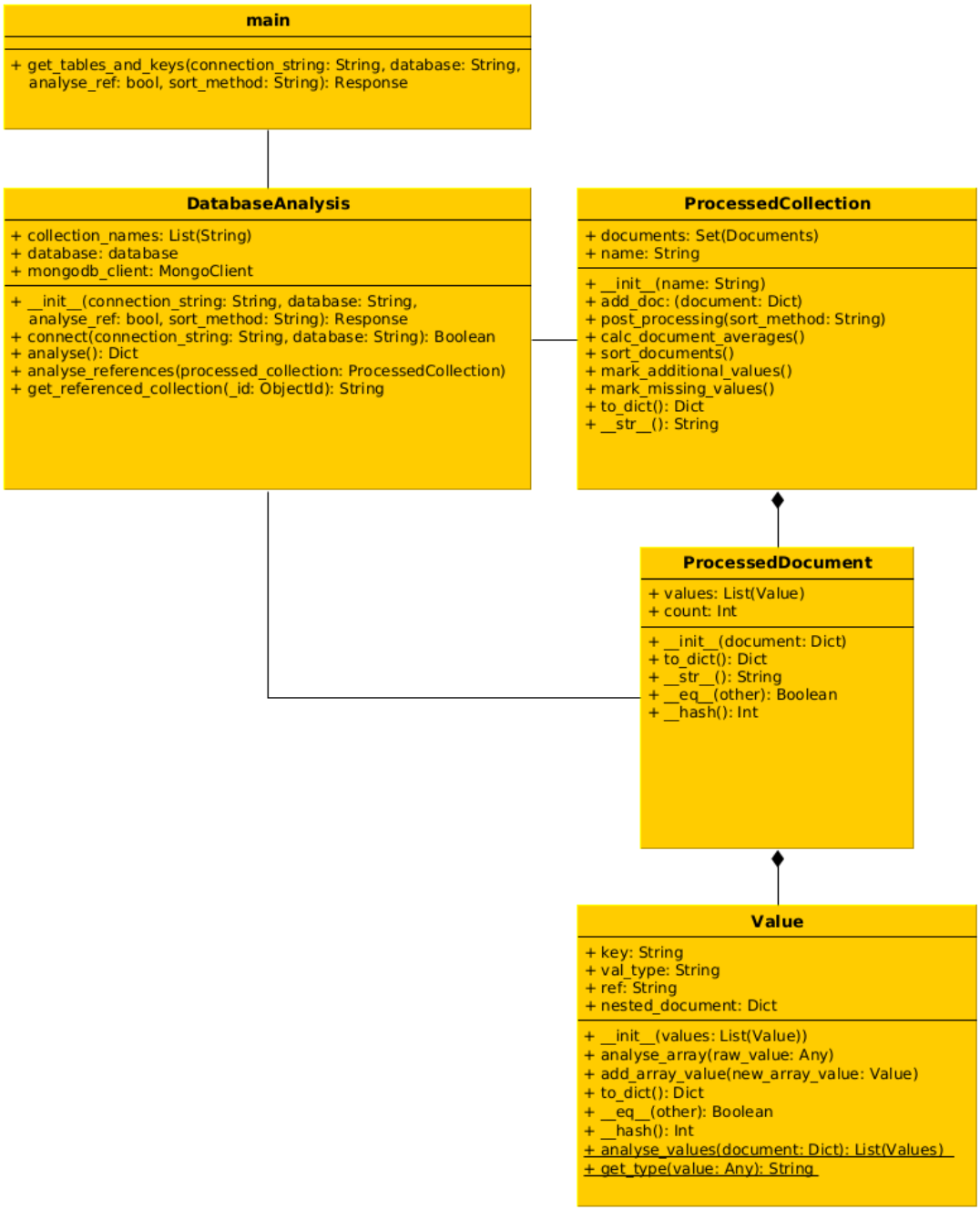

Abbildung 4.4.: Backend UML Diagramm

4.5. Planung des Frontends

Um das Frontend zu planen, wurden in Miro, einem funktionsreichen Online-Whiteboard, Wireframes für die Screens des MongoDB Visualisierungstools gezeichnet.

Um ein einheitliches Layout und damit Anforderung **FA5** zu gewährleisten, wird das gleiche Grundlayout wie in dem ER-Bildschirm des ER Modellierungstools verwendet. Auf der linken Seite gibt es eine Left Sidebar, welche anstatt der ER-Elemente Verbindungsoptionen für die MongoDB Datenbank enthält. Der Hauptscreen in der Mitte zeigt nach erfolgreicher Verbindung die vom Backend extrahierten Hauptschemas jeder Collection als Tabellen an. Diese Tabellen enthalten den Key, den Type und optional die referenzierte Collection. Wenn der Datentyp Array oder Embedded Document ist, kann man eine Untertabelle aufklappen, welche die verschachtelten Daten enthält.

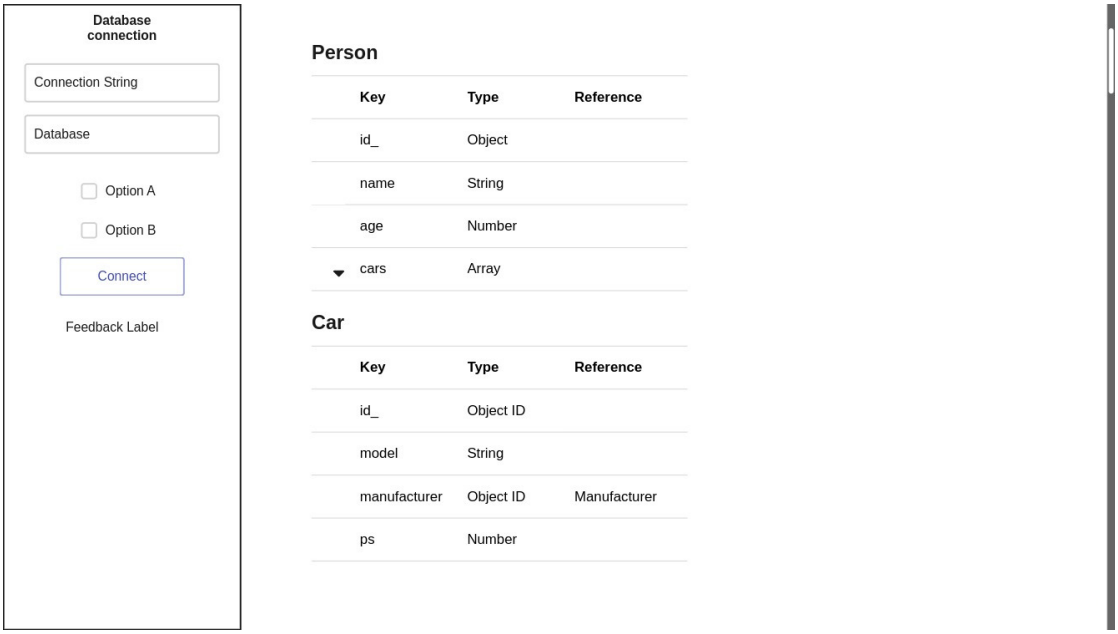

Abbildung 4.5.: Visualization Tool Wireframe

Wenn man auf den Titel einer Tabelle in der Schema Übersicht klickt, öffnet sich die Detail View dieser Collection in einem Popup. In dieser Detail View werden alle Variationen des Schemas angezeigt. Zusätzliche Felder werden grün markiert, und fehlende in rot aufgelistet. Zudem steht über jeder Schemavariation, von wie vielen Dokumenten diese Variation verwendet wird.

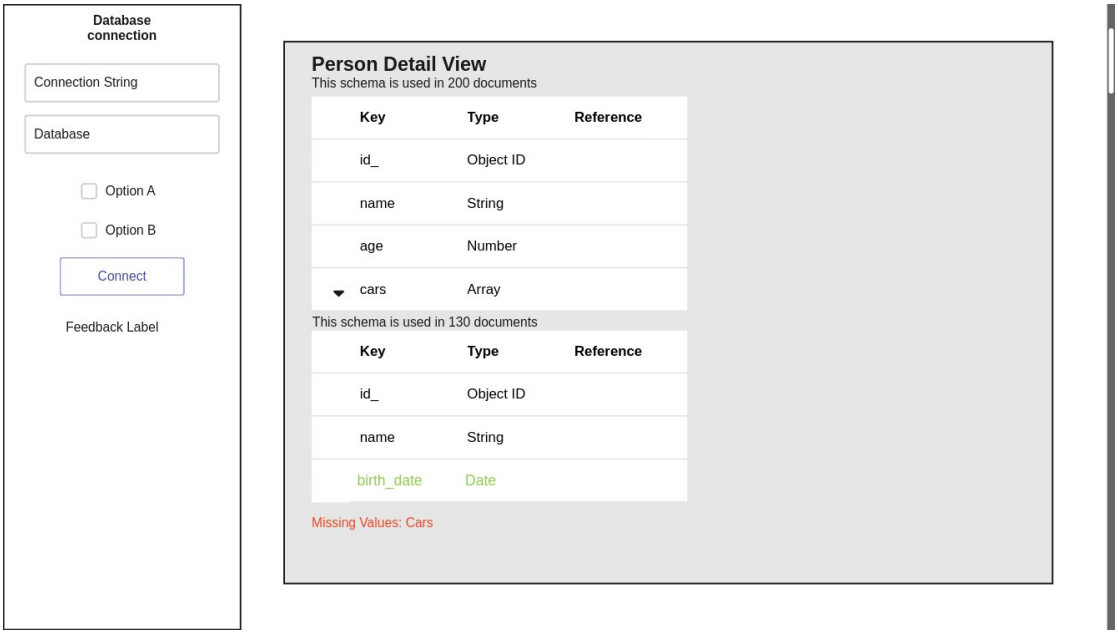

Abbildung 4.6.: Detail View Wireframe

4.6. Modularität des Frontends

Um das Frontend Modular weiterbauen zu können, sind die Elemente im Components Package in ein Package pro Anwendung unterteilt. Alle weiteren Packages werden Anwendungsübergreifend verwendet, sind also nicht weiter unterteilt. Bei einer größeren Anzahl von Anwendungen könnten jedoch auch in diesen Packages Unterteilungen sinnvoll sein.

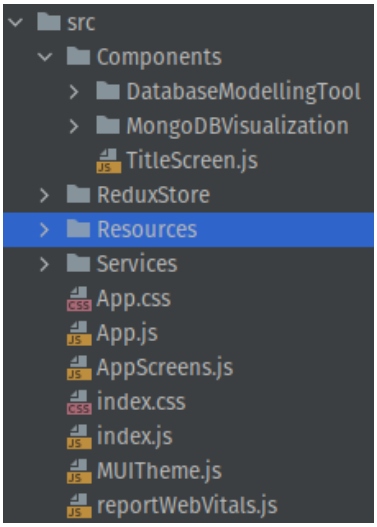

Abbildung 4.7.: Frontend Package Structure

Zudem gibt es einen Startbildschirm welchem aus diese Anwendungen aufgerufen werden können. Um eine neue Anwendung hinzuzufügen, muss man lediglich ein neues Package in Components erstellen und die benötigten Elemente anlegen, sowie im Startbildschirm einen weiteren Button für die Anwendung hinzufügen.

Abbildung 4.8.: Startbildschirm Wireframe

5. Implementierung

Das erarbeitete Lösungskonzept wurde anschließend mit den zuvor beschriebenen Technologien umgesetzt. Die konkrete Implementierung sieht dabei wie folgt aus:

5.1. Backend

5.1.1. Endpunkte

Der bereits beschriebene eine Endpunkt im Backend ließt in Zeile 6 bis 9 alle benötigten Werte aus dem Request Body aus. Mit diesen Daten wird dann ein neues Objekt vom Typ DatabaseAnalysis erstellt. In diesem Objekt wird anschließend die Funktion connect aufgerufen, welche Versucht, sich mit der spezifizierten MongoDB Datenbank zu verbinden. Entsprechend dem Resultat wird entweder ein Fehlercode zurückgegeben oder die Datenbank analysiert und das Resultat der Analyse im JSON-Format im Request Body zurückgegeben.

```
1  app = Flask("Mongodb Visualization Tool")
2  CORS(app)
3
4  @app.post("/connect")
5  def get_tables_and_keys():
6      connection_string = request.json.get("connection_string")
7      database_name = request.json.get("database")
8      analyse_ref = request.json.get("analyse_ref")
9      sort_method = request.json.get("sort_method")
10
11     db_analysis = DatabaseAnalysis(connection_string,
           database_name, analyse_ref, sort_method)
12     connection_successful = db_analysis.connect()
13     if not connection_successful:
14         return Response(status=406)
15
16     document_dict = db_analysis.analyse()
17
18     return json.dumps(document_dict)
```

Quelltext 5.1: app.py

5.1.2. DatabaseAnalysis

Die Methode connect der Klasse DatabaseAnalysis nutzt den MongoDB Client, um eine Verbindung zu einer MongoDB Datenbank mit dem spezifizierten Connection String und dem Datenbanknamen herzustellen. Wenn die Verbindung nach 5 Sekunden noch nicht steht, wird der Versuch abgebrochen und False zurückgegeben. Ansonsten werden die verbundene Datenbank und der verbundene Client im Objekt gespeichert und True zurückgegeben.

```
1 def connect(self, connection_string, database):
2     try:
3         self.mongodb_client = MongoClient(connection_string,
              serverSelectionTimeoutMS=5000)
4         self.database = self.mongodb_client[database]
5     except pymongo.errors.ServerSelectionTimeoutError:
6         return False
7     return True
```

Quelltext 5.2: DatabaseAnalysis.connect

In der Methode analyse derselben Klasse werden daraufhin alle Collection Namen in der Datenbank ausgelesen. Für jeden Namen wird die Collection mit diesem Namen als Dictionary ausgelesen. Dieses Dictionary enthält wiederum alle Dokumente der Collection. Jedes Dokument wird in der Methdode ProcessedCollection.add_doc analysiert und, falls noch nicht vorhanden, der ProcessedCollection hinzugefügt. Danach werden die ProcessedCollections nachbearbeitet. Dies beinhaltet Schritte wie beispielsweise das sortieren der Dokumente und das Markieren der Abweichungen von dem Hauptschema. Wenn alle Dokumente durchlaufen wurden, wird die Verbindung zur Datenbank geschlossen und die ProcessedCollection wird in ein Dictionary umgewandelt und zurückgegeben.

```
1  def analyse(self):
2      self.collection_names = self.database.list_collection_names()
3      docs_dict = {"collections": []}
4      for name in self.collection_names:
5          processed_collection = ProcessedCollection(name)
6          collection = self.database.get_collection(name)
7          documents = collection.find({})
8          for document in documents:
9              processed_collection.add_doc(document)
10
11         processed_collection
               .post_processing(sort_method=self.sort_method)
12         if self.analyse_ref:
13             self.analyse_references(processed_collection)
14         docs_dict["collections"]
               .append(processed_collection.to_dict())
15     self.mongodb_client.close()
```

```
16      return docs_dict
```

Quelltext 5.3: DatabaseAnalysis.analyse

Die Methode analyse_references in DatabaseAnalysis wird nur ausgeführt, wenn der im Endpunkt übergebene Wert analyse_ref auf True gesetzt ist. In der Methode selbst werden alle Values aller verarbeiteten Dokumente durchlaufen und überprüft, ob es sich dabei um eine Referenz auf ein anderes Dokument handeln könnte. Wenn der Datentyp des Values Object ID ist und der Name des Values nicht _id entspricht (Es sich also nicht um den Primary Key handelt), wird angenommen, dass der Value eine Referenz ist. Wenn dies der Fall ist, werden aus den Originaldokumenten, die im Verarbeiteten Dokument abgespeichert sind, die Werte der soeben identifizierten Referenz ausgelesen. Mit diesen Werten wird dann get_referenced_collection so lange aufgerufen, bis die Methode einmal nicht None zurückgibt. get_referenced_collection durchläuft die _id Werte aller (unverarbeiteten) Dokumente und vergleicht diese mit der übergebenen Object ID. Sobald eine Übereinstimmung gefunden wird, wird der Name der Collection zurückgegeben, in der sich das Dokument mit der passenden _id befindet. Wenn keine Übereinstimmung gefunden wird, dann wird None zurückgegeben.

```
1  def analyse_references(self, processed_collection):
2      for document in processed_collection.documents:
3          for value in document.values:
4              if value.val_type == "Object ID" and value.key !=
                   "_id":
5                  for orig_document in document.original_documents:
6                      referenced_collection =
                           self.get_referenced_collection
                           (orig_document.get(value.key))
7                      if referenced_collection is not None:
8                          value.ref = referenced_collection
9                          return
10
11 def get_referenced_collection(self, _id):
12     for collection_name in self.collection_names:
13         collection = self.database.get_collection(collection_name)
14         document = collection.find_one({"_id": _id})
15         if document is not None:
16             return collection_name
17     return None
```

Quelltext 5.4: DatabaseAnalysis.analyse_references

Die Bestimmung der Referenzen mit dieser Methode ist nicht immer akkurat, da der Primary Key eines Schemas auch einen anderen Datentyp als Object ID haben kann. Da die Bestimmung der Referenzen sonst jedoch sehr komplex und rechenintensiv werden würde, werden die Referenzen in dieser Arbeit nur mit dieser vereinfachten

Methode analysiert.

5.1.3. ProcessedCollection

Die Methode add_doc in der Klasse ProcessedCollection erstellt aus dem übergebenen Dictionary ein neues Objekt der Klasse ProcessedDocument. Wenn sich noch kein Objekt mit den gleichen Werten in der aktuellen ProcessedCollection befindet, wird das ProcessedDocument der ProcessedCollection hinzugefügt. Ansonsten wird das Attribut count in dem bereits vorhandenen ProcessedDocument Objekt hochgezählt.

```
1  def add_doc(self, document):
2      new_doc = ProcessedDocument(document)
3      for doc in self.documents:
4          if doc == new_doc:
5              doc.count += 1
6              if doc.document_ages:
7                  doc.document_ages
                       .append(new_doc.document_ages[0])
8              doc.original_documents
                   .append(new_doc.original_documents[0])
9              return
10     self.documents.append(new_doc)
```

Quelltext 5.5: ProcessedCollection.add_doc

Darüber hinaus enthält ProcessedCollection die methode post_processing. post_processing ruft Methoden auf, welche die ProcessedDocuments der ProcessedCollection nachbearbeiten. Folgende Nachbearbeitungsschritte werden ausgeführt:

1. Durchschnittliches Alter aller Dokumente eines ProcessedDocuments ausrechnen
2. Dokumente nach dem Durchschnittsalter oder nach der Anzahl der Aufrufe sortieren
3. Zusätzliche Werte gegenüber dem Hauptschema ermitteln (das Hauptschema ist das Schema, welches in der Sortierung an oberster Stelle steht)
4. Fehlende Werte gegenüber dem Hauptschema ermitteln

```
1  def post_processing(self, sort_method):
2      self.calc_document_averages()
3      self.sort_documents(sort_method)
4      self.mark_additional_values()
5      self.mark_missing_values()
6
```

```
7  def calc_document_averages(self):
8      for document in self.documents:
9          if document.document_ages:
10             document.avg_age = sum(document.document_ages) /
                   len(document.document_ages)
11
12 def sort_documents(self, sort_method):
13     if sort_method == "documentCount":
14         self.documents = sorted(self.documents, key=lambda
               document: document.count, reverse=True)
15     elif sort_method == "avgAge":
16         self.documents = sorted(self.documents, key=lambda
               document: document.avg_age, reverse=True)
17
18 def mark_additional_values(self):
19     main_doc = self.documents[0]
20     for i in range(1, len(self.documents)):
21         for value in self.documents[i].values:
22             if value not in main_doc.values:
23                 value.is_additional = True
24
25 def mark_missing_values(self):
26     main_doc = self.documents[0]
27     for i in range(1, len(self.documents)):
28         for value in main_doc.values:
29             if value not in self.documents[i].values:
30                 self.documents[i]
                       .missing_values.append(value.key)
```

Quelltext 5.6: ProcessedCollection.post_processing

5.1.4. ProcessedDocument

Sämtliche logik in der ProcessedDocument Klasse steckt im Konstruktor. Der Konstruktor nimmt ein Dokument im JSON Format entgegen. In diesem Dokument werden mithilfe der Methode analyse_values in der Datei value.py alle Werte analysiert und die ergebnisse als Liste des eigenen Datentyps Value im ProcessedDocument gespeichert. Wenn der Primary Key vom Typ Object ID ist, wird in Zeile 6 bis 8 das alter des dokuments ermittelt und in einer Liste gespeichert. Diese Liste dient später dazu, das Durchschnittsalter des ProcessedDocuments zu berechnen. Das bestimmen des Alters eines Dokuments ist nur möglich, wenn der Primary Key vom Typ Object ID ist, da Object IDs einen Zeitstempel beinhalten. Des weiteren initialisiert der Konstruktor noch einige Werte, die für die Nachbearbeitung benötigt werden.

```
1  def __init__(self, document):
```

```
2       self.values = analyse_values(document)
3       self.count = 1
4       self.document_ages = list()
5       _id = document.get("_id")
6       if isinstance(_id, bson.ObjectId):
7           document_generation_time =
                document.get("_id").generation_time
8           document_age =
                (datetime.datetime.now(datetime.timezone.utc) -
                document_generation_time).total_seconds()
9           self.document_ages.append(document_age)
10      self.avg_age = None
11      self.missing_values = list()
12      self.original_documents = list()
13      self.original_documents.append(document)
```

Quelltext 5.7: ProcessedDocument.__init__

5.1.5. Value

Die Methode analyse_values in value.py ist eine factory methode, welche für alle Key Value Paare im übergebenen Dokument-Dictionary einen Value erstellt und als Liste zurückgibt.

```
1   def analyse_values(document):
2       values = list()
3       for key, value in zip(document, document.values()):
4           value = Value(key, value)
5           values.append(value)
6       return values
```

Quelltext 5.8: Value.analyse_values

Der Konstruktor der Value Klasse initialisiert alle im ER-Diagramm beschriebenen Values und analysiert den Datentyp des Values mit der Methode get_type. Darüber hinaus wird im Konstruktor überprüft, ob es sich bei dem Value um ein verschachteltes Dokument oder ein Array handelt. Wenn dies der Fall ist, Werden die entsprechenden Analysemethoden aufgerufen.

```
1   def __init__(self, key, raw_value):
2       self.key = key
3       self.ref = None
4       self.nested_document = None
5       self.array_values = None
6       self.val_type = get_type(raw_value)
7       self.is_additional = False
8       if self.val_type == "Embedded document":
```

```
9           self.nested_document = analyse_values(raw_value)
10      elif self.val_type == "Array":
11          self.analyse_array(raw_value)
```

Quelltext 5.9: Value.__init__

get_type enthält simple If-Bedingungen, welche Datentypen eines Werts zu Strings mappen. Wenn der Datentyp nicht von den If-Bedingungen abgedeckt wird, wird eine Exception geworfen.

```
1  def get_type(value):
2      if value is None or type(value) is None:
3          return "Null"
4      if isinstance(value, str):
5          return "String"
6      if isinstance(value, bson.ObjectId):
7          return "Object ID"
8      if isinstance(value, bool):
9          return "Boolean"
10     if isinstance(value, (int, float, complex, bson.Decimal128)):
11         return "Number"
12     if isinstance(value, datetime.date):
13         return "Date"
14     if isinstance(value, collections.abc.Sequence):
15         return "Array"
16     if isinstance(value, dict):
17         return "Embedded document"
18     raise Exception(f"Type {type(value)} of value {value} is not
           identifiable!")
```

Quelltext 5.10: Value.get_type

Wenn es sich bei dem Value um ein Array handelt, wird für jeden Wert im Array ein Value-Objekt erstellt. Die Besonderheit bei diesen Value-Objekten ist, dass das Feld Key leer bleibt, da Values in Arrays keinen eigenen Key besitzen. Wenn das Array array_values des darüberliegenden Values noch keinen Value enthält, der den gleichen Datentyp hat, wird das eben erzeugte Value-Objekt in array_values gespeichert.

```
1  def analyse_array(self, raw_value):
2      self.array_values = list()
3      for arr_element in raw_value:
4          array_value = Value(None, arr_element)
5          self.add_array_value(array_value)
6
7  def add_array_value(self, new_array_value):
8      for array_value in self.array_values:
9          if array_value == new_array_value:
10             return
```

```
11      self.array_values.append(new_array_value)
```

Quelltext 5.11: Value.analyse_array

5.2. Frontend

5.2.1. Startbildschirm und Auswahl des Tools

Die Gesamtlösung des Frontends besteht aus mehreren Tools, die zu einer Toolbox für Datenbanken zusammengefasst werden. Um auswählen zu können, welches Tool man benutzen will, wurde ein Startbildschirm eingefügt. Der Startbildschirm ist ein simpes div, welches den Titel der Anwendung sowie Buttons beinhaltet. Die Buttons sind mit den Namen der einzelnen Tools beschriftet. Die onClick Funktionen der Buttons rufen die Funktion switchAppMode auf, welche als Parameter das enum AppScreen übergeben bekommt. AppScreen teilt der Funktion wiederum mit, welches Tool ausgewählt wurde und geladen werden muss. switchAppMode nutzt dann React Router, um zu der entsprechenden URL des Screens zu navigieren.

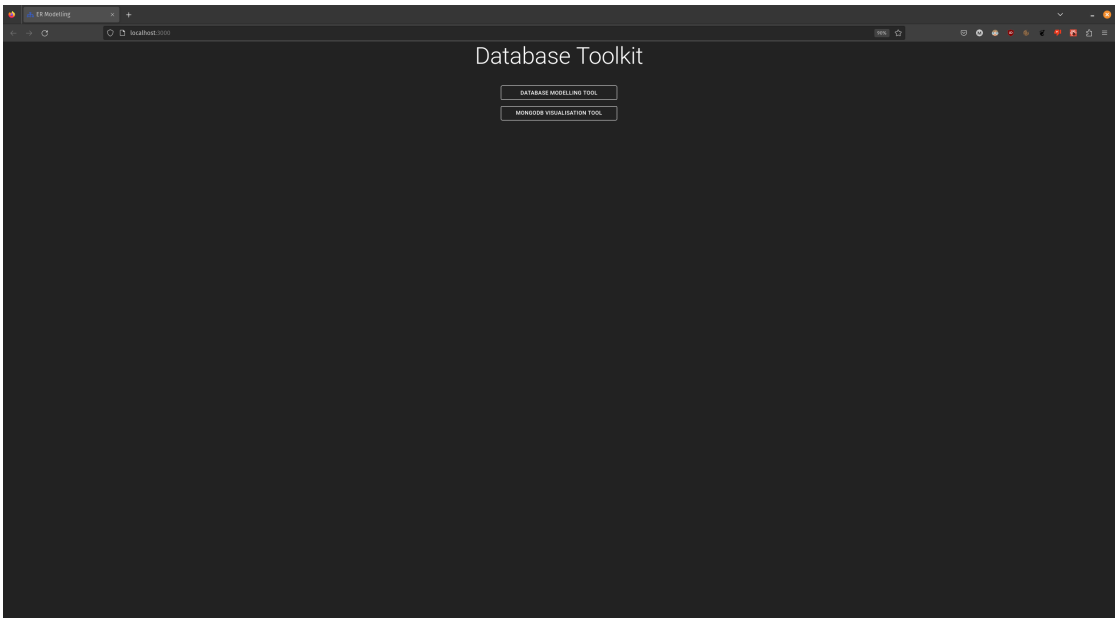

Abbildung 5.1.: Frontend Startbildschrim

Das Rendern der einzelnen Tool Screens erfolgt daraufhin in der Datei App.js. Hier wird mittels React Router der Startbildschirm als Index Element definiert, welches initial angezeigt wird. Zudem wird der Pfad aller weiteren Screens definiert, über welchen diese dann geladen werden können.

```
1 <BrowserRouter>
2     <Routes>
3         <Route index element={<TitleScreen/>}/>
4         <Route path={AppScreens.dbModellingTool}
              element={<DatabaseModellingTool/>}/>
5         <Route path={AppScreens.mongoVisTool}
              element={<MongoManager/>}/>
6     </Routes>
7 </BrowserRouter>
```

Quelltext 5.12: React Router in App.js

5.2.2. Aufbau des MongoDB Visualisation Tool Frontends

Das oberste Element des MongoDB Visualization Tools ist der MongoManager, welcher die MongoLeftSidebar und das MongoDiagram enthält. Mithilfe der MongoLeftSideBar kann die Verbindung und Analyse der MongoDB Datenbank konfiguriert und durchgeführt werden. Nach dem Verbinden und Analysieren werden in dem MongoManager DocumentTables angezeigt, welche rekursiv geschachtelt wieder DocumentTables enthalten können. Darüber hinaus gibt es noch ein DetailView Popup, welches angezeigt wird, wenn man auf den Titel eines DocumentTables klickt. Die DetailView enthält wiederum DocumentTables, welche auch wieder rekursiv geschachtelt sein können. (Das DetailView Popup ist in der nachfolgenden Grafik nicht abgebildet)

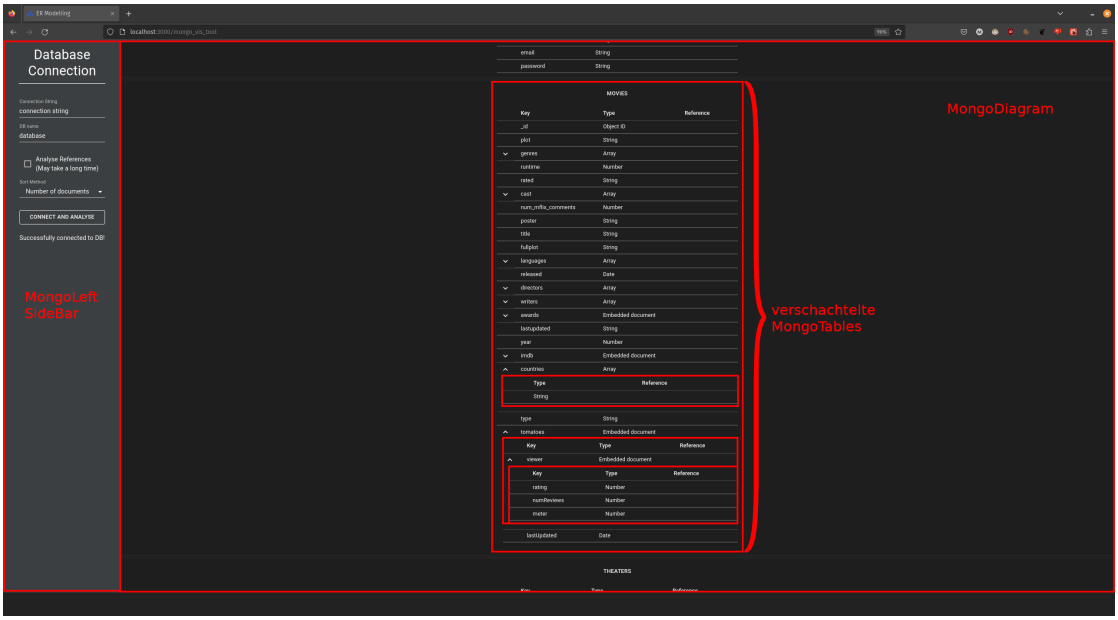

Abbildung 5.2.: Frontend Elemente

5.2.3. Left Sidebar

Die MongoLeftSideBar nutzt die Material UI Komponenten Typography, TextField, FormControl und Button, um das Eingeben der Verbindungs- und Analysedetails zu ermöglichen. Diese Informationen werden bei jeder Änderung im State der MongoLeftSideBar gespeichert. Nach dem Drücken des Connect-Buttons wird die Methode connectToDB aufgerufen. In dieser Methode wird mit den im State gespeicherten Informationen mithilfe von Axios ein HTTP Request an das Backend gesendet. Wenn die Verbindung erfolgreich ist, werden daraufhin die analysierten Daten des Backends im ReduxStore gespeichert. Dies löst daraufhin das Rendering der DocumentTables im MongoDiagram aus. Zudem wird das Feedback Label in der MongoLeftSideBar aktualisiert. Im Fehlerfall wird nur das Feedback Label aktualisiert.

```
1  function connectToDB() {
2      console.log("connecting to db")
3      setConnectionState(ConnectionStates.connecting)
4      const url = "http://127.0.0.1:5000/connect"
5
6      let contentToSend = {
7          connection_string: connectionString,
8          database: dbName,
9          analyse_ref: analyseRef,
10         sort_method: sortMethod
```

```
11      };
12
13      axios.post(url, contentToSend).then((response) => {
14          connectionSuccessful(response)
15          dispatch(setCollections(response.data))
16      }).catch(error => connectionFailed(error))
17  }
18
19  function connectionSuccessful(response) {
20      console.log("data: " + response.data)
21      setConnectionState(ConnectionStates.connected)
22      dispatch(setCollections(response.data))
23  }
24
25  function connectionFailed(error) {
26      console.log(error)
27      setConnectionState(ConnectionStates.connectionFailed)
28  }
```

Quelltext 5.13: MongoLeftSideBar.connectToDB

5.2.4. DocumentTable

Ein DocumentTable repräsentiert ein Schema von Dokumenten ein einer Collection. Es gibt mehrere Arten von DocumentTables, welche je nach Verwendungszweck mit dem Enum DocumentTableType unterschieden werden:

- **main** ist die standard Tabelle, welche im MongoDiagram benutzt wird, um das Hauptschema einer Collection darzustellen.
- **nested** ist eine veschachtelte Untertabelle, entspricht also dem Datentyp Embedded Document.
- **array** wird für den Datentyp Array benutzt.
- **detail** wird im Detail View Popup benutzt, um alle Variatonen der Schemas einer Collection darzustellen.

Dem Element DocumentTable liegt die Material UI Table zugrunde. Material UI Tables haben eine Toolbar, welche hier je nach DocumentTableType angepasst wird. In der Main DocumentTable besteht die Toolbar aus einem Button, mit dem Namen der dargestellten Collection als Label. Die onClick Methode dieses Buttons öffnet die Detail View dieser Collection. Wenn der DocumentTableType detail entspricht, zeigt die Toolbar die Anzahl der Dokumente, die dieses Schema benutzen, sowie eventuell die fehlenden Felder gegenüber dem Hauptschema. In allen anderen Fällen bleibt die Toolbar leer.

Die Spaltenüberschriften im Tablehead sind <Leer> - Key - Type - Reference. Wenn es sich um ein Array Handelt, wird die Spalte Key weggelassen. Die Reihen selbst haben in der ersten Spalte entweder einen Expand Button, mit dem sich Untertabellen und Arrays ausklappen lassen, oder nichts. In den nachfolgenden Spalten werden die Daten aus dem Backend eingefüllt. Nach jeder Reihe folgt eine zunächst unsichtbare Reihe. Wenn der Datentyp des Werts in dieser Reihe Array oder Embedded document entspricht, wird in dieser unsichtbaren Reihe die entsprechende Untertabelle geladen. Diese lässt sich dann mit dem Expand Button ausklappen und anzeigen.

5.2.5. DetailView Popup

Tatsächlich rendert die return Methode des DetailViewPopups zunächst nur einen Button. Erst wenn dieser Button gedrückt wird, wird das Popup als Material UI Modal angezeigt. Dieses Modal lädt daraufhin, wie auch der MongoManager, DocumentTables. Der DocumentTableType ist in diesen Tabellen jedoch nicht main, sondern detail, was die Toolbars der Tabellen verändert.

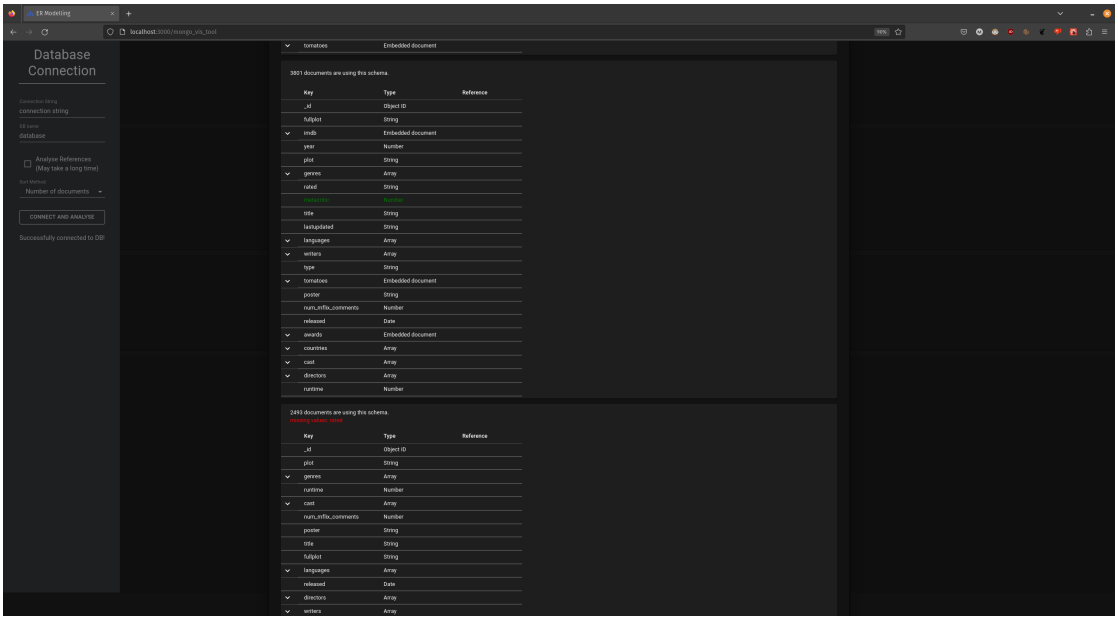

Abbildung 5.3.: Frontend Detail View Popup

5.2.6. Redux Store

Der Redux Store dient dem Speichern von States losgelöst von einzelnen React Komponenten. Für das Speichern eines States wird ein sogenannter ContentSlice

verwendet. Die ContentSlices für das MongoDB Visualisation Tool sind grundsätzlich alle gleich aufgebaut: Im initialState wird der Ausgangszustand des States definiert. Im MongoContentSlice wird initial collections auf null gesetzt. Reducer sind Methoden, welche das Beschreiben des States erlauben. in diesem Fall wurde der Reducer setCollections definiert, welcher es erlaubt, in collections das argument documents zu schreiben. Abschließend werden die actions und die reducer des ContentSlice exportiert, sodass diese global erreichbar sind.

```
export const mongoContentSlice = createSlice({
    name: 'mongoContent',
    initialState: {
        collections: null
    },
    reducers: {
        setCollections: (state, documents) => {
            state.collections = documents
        }
    }
})

export const {setCollections} = mongoContentSlice.actions
export default mongoContentSlice.reducer
```

Quelltext 5.14: MongoContentSlice

Auf diese Weise lassen sich schnell und einfach weitere ContentSlices definieren. Um diese ContentSlices zu verwenden, müssen sie nur noch in den Redux Store eingefügt werden.

```
export const store = configureStore({
    reducer: {
        erContent: erContentSlice,
        relationalContent: relationalContentSlice,
        mongoContent: mongoContentSlice
    }
})

export default store;
```

Quelltext 5.15: Redux Store

6. Inbetriebnahme

Das MongoDB Visualisation Tool wird auf einem Linux Server mit Debian 11 gehostet. Das Flask Backend läuft in einem Docker Container. Der Vorteil von Docker ist, dass der Applikation eine eigene Umgebung bereitgestellt wird, die unabhängig vom Host-System ist. Um das Backend als Dockercontainer auszuliefern, wurde das vorkonfigurierte Docker Image uwsgi-nginx-flask von tiangolo genutzt. In diesem Docker Image sind uWSGI und Nginx vorinstalliert, was das ausliefern von Flask Applikationen erleichtert. [19]

Das Backend für das ER Modelling Tool ist als Heroku-App deployed. Heroku ist ein cloudbasierter Serviceanbieter, welcher ein kostenloses Modell für das Hosting von Software und DNS anbietet. [7]

Das Frontend wurde mittels NPM gebaut und auf dem Server bereitgestellt. Mittels NGINX wurde das Webinterface daraufhin gehostet. Das Interface ist nun unter der Adresse `http://kodewisch.com:12030` erreichbar.

6.1. Buildprozess des Backends

Um das Backend starten zu können, muss Python 3.10 oder neuer installiert sein.

Wenn dies der Fall ist, können mit dem Befehl `pip -r requirements.txt` im Verzeichnis mongo_vis_backend alle benötigten Pakete installiert werden. Um Konflikte mit anderen Python installationen zu vermeiden, führt man diesen Befehl am besten in einem Virtual Environment aus. Anschließend kann das Backend mit dem Befehl `python3 -m flask run` ausgeführt werden.

Eine Anleitung für das Deployment des Backends als Docker Container kann unter `https://hub.docker.com/r/tiangolo/uwsgi-nginx-flask/` gefunden werden.

6.2. Buildprozess des Frontends

Node und NPM müssen installiert sein, damit das Frontend gebaut werden kann. Über tps://nodejs.org/en/download/ können Node und NPM installiert werden.

Anschließend müssen im Verzeichnis frontend folgende Befehle ausgeführt werden, um es zu starten:

```
$ npm install
$ npm start
```

Quelltext 6.1: Frontend build commands

Um das Frontend auszuliefern, kann das Frontend mit dem Befehl `npm run build` gebaut werden. Die gebauten Dateien befinden sich dann im Ordner build.

Vorausgesetzt wird ein Debian- oder Ubuntubasierter Computer/ Server, auf dem Nginx mit SSL Zertifikat konfiguriert ist. Um die Dateien auf solch einem Server auszuliefern, müssen die Dateien in den Ordner /var/www/db-toolkit/html verschoben werden. db-toolkit kann auch durch einen beliebigen Namen ersetzt werden. Daraufhin muss folgende Nginx Konfiguration im Ordner /etx/nginx/conf.d mit dem namen db-toolkit.conf angelegt werden:

```
server {
    listen 80;
    listen [::]:80;
    server_name db-toolkit.kodewisch.com;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name db-toolkit.kodewisch.com;

    ssl_certificate
       /etc/letsencrypt/live/kodewisch.com/fullchain.pem;
    ssl_certificate_key
       /etc/letsencrypt/live/kodewisch.com/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    root /var/www/db-toolkit/html;
    index index.html index.htm;

    location / {
        try_files $uri $uri/ =404;
    }
```

```
23 }
```

Quelltext 6.2: db-toolkit.conf

Diese Nginx Konfiguration zeigt die mit root und index spezifizierte Resource für alle anfragen an, die auf die Subdomain db-toolkit.kodewisch.com eingehen. Die obere Server-Konfiguration nimmt die HTTP Anfragen auf Port 80 entgegen und leitet diese auf HTTPS um. Die untere Server-Konfiguration nimmt die HTTPS Anfragen auf Port 443 entgegen, spezifiziert das SSL-Zertifikat und zeigt die Resource an. Die ssl_certificate Pfade müssen durch die entsprechenden Pfade auf dem Server ersetzt werden, ebenso wie die Domain.

7. Evaluierung

Die Anforderung **FA1** verlangt, dass das MongoDB Visualisation Tool in einer gemeinsamen Webapp mit dem ER Modelling Tool implementiert wird. Dies wurde durch das Einfügen eines Startbildschirms zur Auswahl des Tools erreicht (**FA3**). Der Startbildschirm erleichtert zudem das Einfügen weiterer Tools in das Database Toolkit. **FA2** verlangt, dass die Anwendung modular ist, um einfach weitere Tools hinzuzufügen. Diese Anforderung wird neben dem Startbildschirm durch die verbesserte Package Struktur im Frontend, sowie der Microservice Architektur im Backend erfüllt.

FA4 verlangt, dass das Frontend intuitiv Benutzbar ist. Die Interaktionsmöglichkeiten mit dem MongoDB Visualisation Tool sind relativ klein, da die Hauptaufgabe der Anwendung Visualisierung ist. Der Großteil der Interaktionsmöglichkeiten befindet sich in der LeftSideBar. In dieser werden nur Standardkomponenten von Material UI verwendet, deren Benutzung jedem Nutzer geläufig sind. Zudem sind alle Elemente aussagekräftig beschriftet. Deshalb ist die Anforderung **FA4** erfüllt.

Um die Intuitive Benutzbarkeit zu gewährleisten, müssen alle Tools ein einheitliches Design und Layout besitzen (**FA5**). Das einheitliche Layout wird durch Verwendung der gleichen Grundelemente wie das ER Modelling Tool gewährleistet. Um das einheitliche Design sicherzustellenbn, wurden die gleichen Farben, Schriftgrößen und Co. in den CSS Dateien verwendet. Um Entwicklern weiterer Tools das einhalten eines einheitlichen Designs zu erleichtern, wurde ein Material UI Theme erstellt, und wo möglich, Material UI Komponenten verwendet.

Die übersichtliche Darstellung der Schemata (**FA6**) wurde durch kustomisierte Material UI Tabellen realisert. Durch das dynamische Ausklappen von verschachtelten Dokumenten sowie Arrays lassen sich beliebig tief verschachtelte Schemata übersichtlich darstellen. Damit die Unterschiede aller Schemata in einer Collection übersichtlich dargestellt werden, gibt es eine Detail View, in der alle Abweichungen des Hauptschemas Farblich markiert sind. Dadurch ist die Anforderung **FA7** erfüllt.

Eine Anforderung an das Backend ist, dass auch große Datenbanken in wenigen Sekunden analysiert werden können müssen (**BA1**). Um die Erfüllung dieser Anforderung zu verifizieren, wurden mit Test MongoDB Datenbanken Tests durchgeführt. Als Test Datenbanken wurden die Sample Datenbanken von MongoDB Atlas genutzt, welche auch über MongoDB Atlas gehostet werden. Für diesen Test wurde das Backend lokal auf einem Laptop mit dem Prozessor AMD Ryzen 5 4500U (6

Kerne) gehostet. Jede Datenbank wurde mehrfach ohne und mit Referenzanalyse getestet. Durch das mehrfache Testen werden Testungenauigkeiten vermieden.

Diese Tests ergaben folgende Resultate:

Datenbank Name	Anzahl Collections	Anzahl Dokumente	Datenmenge	Analyse-Dauer	Dauer (mit Referenzen)
sample_airbnb	1	5555	89.99MB	10.8s	10.9
sample_analytics	3	3992	15.79MB	2.7s	2.7s
sample_geospatial	1	11095	3.47MB	1.1s	1.1s
sample_guides	1	8	1.29KB	0.3s	0.3s
sample_mflix	5	66359	47.57MB	8.3s	9.0s
sample_restaurants	2	25554	13.36MB	3.5s	3.5s
sample_supplies	1	5000	4.13MB	1.3s	1.3s
sample_training	7	296502	113.76MB	29.9s	27.0s
sample_weatherdata	1	10000	16.15MB	4.0s	3.9s

Die Analyse der meisten Datenbanken bewegt sich in einem Zeitraum von unter 11 Sekunden. Dies ist für die Nutzer eine annehmbare Zeit, da den Nutzern durch ein Fortschrittsrad signalisiert wird, dass Berechnungen vorgenommen werden. Dadurch bekommen die Nutzer nicht das Gefühl, dass die Anwendung sich aufgehangen hätte. Einzig die Analyse der Datenbank sample_training mit fast 300.000 Dokumenten dauert circa 30 Sekunden. Die Anforderung **BA1** ist also nur für Datenbanken bis zu einer gewissen Größe erfüllt. Dies ist also ein Punkt, in dem das MongoDB Visualisierungstool noch Verbesserungsbedarf hat.

Der Endpunkt im Flask Backend ist von mehreren Nutzern gleichzeitig ansteuerbar. Damit es beim Verbinden und Analysieren mehrerer Datenbanken gleichzeitig nicht zu Konflikten kommt, ist der gesamte Prozess Objektorientiert aufgebaut. Dadurch hat jede Analyse ihre eigenen Objekte, die nicht mit anderen interferieren. Die Verbindung selbst erfolgt über die Bibliothek Pymongo, welche das dynamische Anbinden beliebig vieler MongoDB Datenbanken erlaubt. Dadurch ist die Anforderung **BA2** erfüllt.

8. Zusammenfassung und Ausblick

8.1. Erreichte Ergebnisse

In diesem Projekt wurde das Problem angegangen, dass es keine geeigneten Schema-Analysetools für MongoDB Datenbanken gibt. Deshalb wurde das MongoDB Visualisierungstool Dieses Tool wurde in das bestehende Projekt ER Modellierungstool integriert, um ein erweiterbares Datenbank Toolkit zu bilden. Das Tool selbst besteht dabei aus 2 Teilen: Der erste Teil ist ein Frontend, welches in das ER Modellierungstool Frontend integriert wurde. Dieses Frontend wurde mit JavaScript und React umgesetzt. Die Modularität dieser Gesamtlösung wurde durch einen Startbildschirm, sowie strukturelle Anpassungen im Code beibehalten und erweitert. Das MongoDB Visualisierungstool Frontend besteht aus drei Hauptelementen:

- Der LeftSidebar, die das Verbinden mit einer MongoDB Datenbank ermöglicht
- Dem MongoDiagram, welches einen Überblick über die meistverwendeten Schemas in den Collections gibt
- und dem DetailView Popup, welches alle Schemas in einer Collection anzeigt und die Unterschiede hervorhebt.

Die Schemas wurden hierbei mit der Komponente DokumentTable umgesetzt, welche auf der Material UI Table Komponente basiert.

Der zweite Teil des MongoDB Visualisierungstools ist ein Backend, welches in dem Python-Framework Flask geschrieben wurde. Das Backend besitzt einen einzigen REST-Endpunkt. Die Aufgabe dieses Endpunkts ist es, sich mit der spezifizierten Datenbank zu verbinden und diese zu analysieren. Die Ergebnisse der Analyse werden an das Frontend im JSON-Format zurückgegeben welches diese dann visualisert.

8.2. Ausblick

Es gibt ein Paar Funktionen des MongoDB Visualisierungstools, die noch weiter ausgebaut werden könnten.

8.2.1. Referenzen

Einerseits Ist das Ermitteln der Referenzen noch nicht besonders genau. In dem aktuellen Algorithmus wird davon ausgegangen, dass alle Werte des Typs Object ID, welche nicht den namen _id besitzen, Referenzen sind. Jedoch kann es auch Referenzen geben, die nicht den Typ Object ID besitzen, da der Primary Key eines Dokuments nicht zwangsläufig eine Object ID sein muss. Deshalb könnte man die Bestimmung der Referenzen noch weiter verbessern.

Die Visualisierung der Referenzen könnte ebenfalls ausgebaut werden: Man könnte eine Ansicht der Collections als Graph mit den Collections als Knoten und den Referenzen als Kanten darstellen. Dies würde es Entwicklern bei Datenbanken mit vielen Referenzen erleichtern, die Abhängigkeiten zwischen den Collections zu Überblicken.

8.2.2. Performance der Analyse

Wie in dem Experiment im Kapitel Evaluierung festgestellt wurde, kann die Analyse von Größeren Datenbanken lange dauern. Die Analyse der größten Datenbank im Experiment dauerte 30 Sekunden. Solch eine lange Wartezeit unterbricht den Arbeitsablauf eine Entwicklers, und sollte deswegen möglichst minimiert werden, Dafür gibt es mehrere Ansätze:

Mittels Multithreading die Analysezeit erheblich reduziert werden. Beispielsweise könnte jede Collection auf einem eigenen Thread analysiiert werden. Da das Backend in Python geschrieben wurde, ist die Umsetzung von Multithreading relativ simpel. Python bietet mit der Bibliothek MultiProcessing eine simple, aber effiziente Implementierung von Multithreading. [12]

Eine weitere Möglichkeit, die Laufzeit der Analyse zu senken, ist, in größeren Datenbanken optional nur einen Teil der Dokumente jeder Collection zu analysieren. Damit könnte, je nach Anzahl der nicht analysierten Dokumente, die Laufzeit linear gesenkt werden. Jedoch ist es mit dieser Methode nicht mehr möglich, alle Schemavariationen einer Collection zuverlässig anzuzeigen.

Abgesehen von Software-Seitigen Optimierungen kann die Laufzeit verbessert werden, in dem man das Backend auf einem Leistungsstärkeren Server ausliefert.

8.2.3. Weitere Datenbanksysteme

Neben MongoDB gibt es noch eine Vielzahl weiterer Datenbanksysteme, für welche keine geeigneten Visualisierungstools existieren. Man könnte das MongoDB Visua-

lisierungstool beispielsweise zu einem NoSQL Visualisierungstool erweitern. Man könnte dafür in der LeftSideBar den Datenbanktyp auswählen, und daraufhin in dem Hauptbildschirm eine passende Visualisierung der Datenbank generieren. Vor allem für Dokument-Datenbanken und andere zu MongoDB ähnliche Datenbanken wäre der Aufwand, diese zu integrieren, gering, da die nötigen Komponenten zur Visualisierung dieser bereits existieren, und lediglich angepasst werden müssten. Aber auch andere NoSQL Datenbanken, wie beispielsweise Graph-Datenbanken, könnten in das Visualisierungstool integriert werden.

8.2.4. Weitere Tools

Neben den zwei bestehenden Tools kann das Datenbank Toolkit um weitere Tools erweitert werden. Das Datenbank Toolkit ist entsprechend modular aufgebaut, dass weitere Tools ohne große Änderungen in das bestehende Toolkit integriert werden können.

Literatur

[1] Ralf Abueg. *Visualization Solutions For MongoDB*. 2020. URL: https://www.knowi.com/blog/visualization-solutions-for-mongodb/ (besucht am 30. 01. 2023).

[2] Stephan Augsten. *Was sind Container?* 2017. URL: https://www.dev-insider.de/was-sind-container-a-573872/ (besucht am 22. 02. 2023).

[3] Alex Banks und Eve Porcello. *Learning React: Modern Patterns for Developing React Apps*. O'Reilly Media, 2020.

[4] Shannon Bradshaw, Eoin Brazil und Kristina Chodorow. *MongoDB: the definitive guide: powerful and scalable data storage*. O'Reilly Media, 2019.

[5] Soham De Roy. *What is Redux? Store, Actions, and Reducers Explained for Beginners*. 2022. URL: https://www.freecodecamp.org/news/what-is-redux-store-actions-reducers-explained/ (besucht am 20. 01. 2023).

[6] Miguel Grinberg. *Flask web development: developing web applications with python*. Ö'Reilly Media, Inc.", 2018.

[7] *Heroku*. 2023. URL: https://www.heroku.com (besucht am 21. 02. 2023).

[8] *Introducing JSON*. URL: https://www.json.org/json-en.html (besucht am 11. 08. 2022).

[9] Selina Khoirom u. a. „Comparative analysis of Python and Java for beginners". In: *Int. Res. J. Eng. Technol* 7.8 (2020), S. 4384–4407.

[10] *MongoDB Systemeigenschaften*. 2023. URL: https://db-engines.com/de/system/MongoDB (besucht am 30. 01. 2023).

[11] *MUI: The React component library you always wanted*. 2023. URL: https://mui.com/ (besucht am 30. 01. 2023).

[12] *multiprocessing - Process-based parallelism*. 2023. URL: https://www.dev-insider.de/was-sind-container-a-573872/ (besucht am 23. 02. 2023).

[13] *nginx*. 2023. URL: https://nginx.org/en/ (besucht am 21. 02. 2023).

[14] *PyMongo 4.3.3 Documentation*. 2022. URL: https://github.com/mongodb/mongo-python-driver (besucht am 30. 01. 2023).

[15] Edwin Schicker. *Datenbanken und SQL*. 5. Auflage. Regensburg: Springer, 2017. ISBN: 978-3-658-16128-6.

[16] Marcus Schießer und Martin Schmollinger. *Workshop Java EE 7*. 2., aktualisierte und erweiterte Auflage. Heidelberg: dpunkt.verlag, 2015, S. 1–13. ISBN: 978-3-86490-195-9.

[17] *Spring Data MongoDB*. 2023. URL: https://spring.io/projects/spring-data-mongodb (besucht am 30. 01. 2023).

[18] Söderlund Sverker. „Performance of REST applications“. Diss. 2017.

[19] tiangolo. *uwsgi-nginx-flask*. 2023. URL: https://hub.docker.com/r/tiangolo/uwsgi-nginx-flask (besucht am 21. 02. 2023).

[20] Guido Van Rossum und Fred L Drake Jr. *Python tutorial*. Bd. 620. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 2020.

[21] J Sergio Zepeda und Sergio V Chapa. „From desktop applications towards ajax web applications“. In: *2007 4th International Conference on Electrical and Electronics Engineering*. IEEE. 2007, S. 193–196.

A. Anhang A

B. Anhang B