

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

Ziel der Kurzfassung ist es, einen (eiligen) Leser zu informieren, so dass dieser entscheiden kann, ob der Bericht für ihn hilfreich ist oder nicht (neudeutsch: Management Summary). Die Kurzfassung gibt daher eine kurze Darstellung

- des in der Arbeit angegangenen Problems
- der verwendeten Methode(n)
- des in der Arbeit erzielten Fortschritts.

Dabei sollte nicht auf die Struktur der Arbeit eingegangen werden, also Kapitel 2 etc. denn die Kurzfassung soll ja gerade das Wichtigste der Arbeit vermitteln, ohne dass diese gelesen werden muss. Eine Kapitelbezogene Darstellung sollte sich in Kapitel 1 unter Vorgehen befinden.

Länge: Maximal 1 Seite.

# Inhaltsverzeichnis

<b>Eidesstattliche Erklärung</b>	<b>i</b>
<b>Kurzfassung</b>	<b>ii</b>
<b>Inhaltsverzeichnis</b>	<b>iii</b>
<b>Abbildungsverzeichnis</b>	<b>vii</b>
<b>Quelltextverzeichnis</b>	<b>viii</b>
<b>Abkürzungsverzeichnis</b>	<b>ix</b>
<b>1. Einleitung</b>	<b>1</b>
1.1. Motivation . . . . .	1
1.2. Problemstellung und -abgrenzung . . . . .	1
1.3. Vorangegangene Arbeit . . . . .	2
1.4. Ziel der Arbeit . . . . .	2
<b>2. Grundlagen</b>	<b>3</b>
2.1. JSON . . . . .	3
2.2. HTTP . . . . .	5
2.3. REST API . . . . .	5
2.4. SQL . . . . .	5
2.5. MongoDB . . . . .	5

2.6. Python . . . . .	6
2.7. Flask . . . . .	6
2.8. JavaScript . . . . .	6
2.9. React . . . . .	7
2.9.1. Redux Store . . . . .	7
2.9.2. Axios . . . . .	7
<b>3. Problemanalyse</b>	<b>8</b>
3.1. Anforderungen an das Frontend . . . . .	8
3.2. Anforderungen an das Backend . . . . .	9
3.3. Übersicht über die Anforderungen . . . . .	9
<b>4. Lösungskonzept</b>	<b>10</b>
4.1. Bestehende Visualisierungstools . . . . .	10
4.1.1. MongoDB Data Explorer . . . . .	10
4.1.2. MongoDB Compass . . . . .	11
4.1.3. MongoDB Charts, Tableau, Qlik und Looker . . . . .	11
4.2. Verwendete Technologien . . . . .	12
4.2.1. Backend Technologien . . . . .	12
4.2.2. Frontend Technologien . . . . .	13
4.3. REST-Schnittstelle . . . . .	14
4.4. Analyse der MongoDB Datenbank . . . . .	15
4.5. Planung des Frontends . . . . .	16
4.6. Modularität des Frontends . . . . .	17

<b>5. Implementierung</b>	<b>19</b>
5.1. Backend . . . . .	19
5.1.1. Endpunkte . . . . .	19
5.1.2. DatabaseAnalysis . . . . .	20
5.1.3. ProcessedCollection . . . . .	22
5.1.4. ProcessedDocument . . . . .	23
5.1.5. Value . . . . .	24
5.2. Frontend . . . . .	26
5.2.1. Startbildschirm und Auswahl des Tools . . . . .	26
5.2.2. Aufbau des MongoDB Visualisation Tool Frontends . . . . .	27
5.2.3. Left Sidebar . . . . .	28
5.2.4. DocumentTable . . . . .	29
5.2.5. DetailView Popup . . . . .	30
5.2.6. Redux Store . . . . .	30
<b>6. Inbetriebnahme</b>	<b>32</b>
<b>7. Evaluierung</b>	<b>33</b>
<b>8. Zusammenfassung und Ausblick</b>	<b>34</b>
8.1. Erreichte Ergebnisse . . . . .	34
8.2. Ausblick . . . . .	34
8.2.1. Erweiterbarkeit der Ergebnisse . . . . .	34
8.2.2. Übertragbarkeit der Ergebnisse . . . . .	34
<b>Literatur</b>	<b>35</b>
<b>A. Anhang A</b>	<b>37</b>



# Abbildungsverzeichnis

2.1. JSON Object . . . . .	3
2.2. JSON Array . . . . .	3
2.3. JSON Value . . . . .	4
2.4. JSON String . . . . .	4
2.5. JSON Number . . . . .	4
4.1. MongoDB Data Explorer . . . . .	10
4.2. MongoDB Compass . . . . .	11
4.3. MongoDB Charts . . . . .	12
4.4. Backend UML Diagramm . . . . .	15
4.5. Visualization Tool Wireframe . . . . .	16
4.6. Detail View Wireframe . . . . .	17
4.7. Frontend Package Structure . . . . .	18
5.1. Frontend Startbildschirm . . . . .	26
5.2. Frontend Elemente . . . . .	28
5.3. Frontend Detail View Popup . . . . .	30



# Listings

5.1. app.py . . . . .	19
5.2. DatabaseAnalysis.connect . . . . .	20
5.3. DatabaseAnalysis.analyse . . . . .	20
5.4. DatabaseAnalysis.analyse_references . . . . .	21
5.5. ProcessedCollection.add_doc . . . . .	22
5.6. ProcessedCollection.post_processing . . . . .	22
5.7. ProcessedDocument.__init__ . . . . .	24
5.8. Value.analyse_values . . . . .	24
5.9. Value.__init__ . . . . .	24
5.10. Value.get_type . . . . .	25
5.11. Value.analyse_array . . . . .	25
5.12. React Router in App.js . . . . .	27
5.13. MongoLeftSideBar.connectToDB . . . . .	28
5.14. MongoContentSlice . . . . .	31
5.15. Redux Store . . . . .	31

# Abkürzungsverzeichnis

<b>SQL</b> Structured Query Language . . . . .	5
<b>HTTP</b> Hypertext Transfer Protocol . . . . .	5
<b>API</b> Application Programming Interface . . . . .	5
<b>REST</b> Representational State Transfer . . . . .	5
<b>JSON</b> JavaScript Object Notation . . . . .	3
<b>UI</b> User Interface . . . . .	7
<b>DDL</b> Data Definition Language . . . . .	5
<b>DML</b> Data Manipulation Language . . . . .	5
<b>XML</b> Extensible Markup Language . . . . .	5
<b>JAX-RS</b> Jakarta RESTful Web Services . . . . .	5
<b>POJO</b> Plain Old Java Object . . . . .	12

# 1. Einleitung

## 1.1. Motivation

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. [8] Da MongoDB keine relationale Datenbank ist, können herkömmliche Visualisierungs- und Analysetools, die für SQL entwickelt wurden, nicht verwendet werden. Für MongoDB gibt es zwar ein paar 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] Ziel dieses Projekts ist es deshalb, ein Visualisierungstool für MongoDB Datenbanken zu entwickeln, welches die Dokumente der Datenbanken analysiert und auswertet. Diese Schemas werden dann in Form von verschachtelten Tabellen visualisiert. Das erleichtert es den Entwicklern der Datenbanken, die Strukturen und Abhängigkeiten in ihren Datenbanken zu verstehen und zu verbessern.

## 1.2. Problemstellung und -abgrenzung

Um Datenbanken analysieren zu können, muss es möglich sein, sich mit diesen zu verbinden. Dies erfordert einerseits eine Benutzeroberfläche, über die der Nutzer die Verbindungsdaten eingeben kann. Andererseits erfordert dies die Verbindung mit der Datenbank selbst über eine geeignete MongoDB Schnittstelle. 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.

### **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**

Das Ziel der Projektarbeit ist die Planung und Umsetzung der oben beschriebenen Problemstellung. 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 dafür in die vorangegangene Arbeit integriert werden und die Gesamtlösung dabei modular halten.

## 2. Grundlagen

### 2.1. JSON

JavaScript Object Notation (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: [6]

**Object** ist ein Set aus Key/Value Paaren.

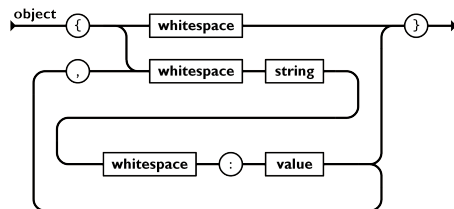


Abbildung 2.1.: JSON Object

**Array** ist eine geordnete Liste von Values.

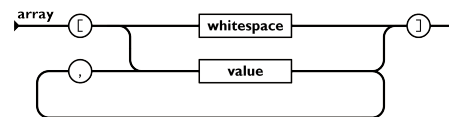


Abbildung 2.2.: JSON Array

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

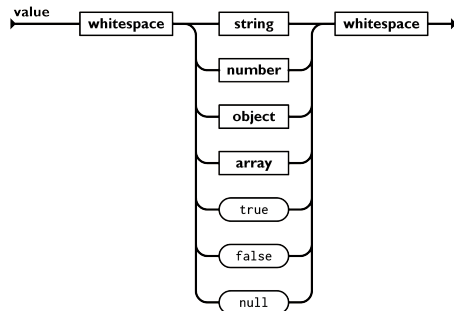


Abbildung 2.3.: JSON Value

**String** ist eine Sequenz aus Unicode Buchstaben.

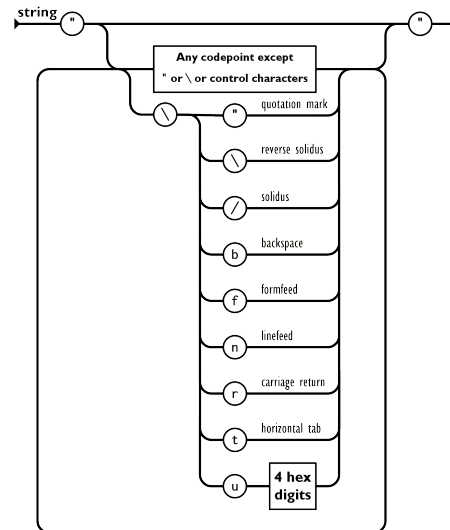


Abbildung 2.4.: JSON String

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

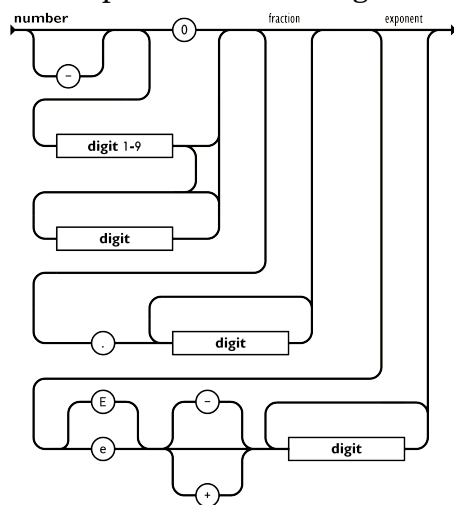


Abbildung 2.5.: JSON Number

## 2.2. HTTP

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

## 2.3. REST API

Ein Application Programming Interface (API) ist eine Schnittstelle, über die von außen mit einem Programm interagiert werden kann. Das Grundprinzip von Representational State Transfer (REST) ist ein Backend, welches Ressourcen beinhaltet. Ein Client kann über die gängigen HTTP Operationen mit diesen Ressourcen interagieren. Jede Resource hat eine global eindeutige ID und wird meist durch JSON oder Extensible Markup Language (XML) repräsentiert. In Java wird ein RESTful Webservice für gewöhnlich mit Jakarta RESTful Web Services (JAX-RS) umgesetzt. [\[12\]](#)

## 2.4. 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. [\[11\]](#)

## 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, das 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. [3]

## 2.6. Python

Python ist eine Objektorientierte High-Level Programmiersprache, die besonderen Wert auf Lesbarkeit legt. Variablen in Python werden dynamisch getyped. 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. [16]

## 2.7. Flask

Flask ist ein minimalistisches Open-Source Web Framework. Flask kommt mit ein paar Kernpaketen, welche für Web Apps benötigt werden. Alles weitere muss der Nutzer selbst über PIP installieren. Durch diesen minimalistischen Ansatz kann man Flask sehr flexibel einsetzen. Flask ist beispielsweise flexibel mit SQL oder NoSQL Datenbanken, aber auch ohne Datenbanken einsetzbar. [5]

## 2.8. JavaScript

JavaScript ist eine High-Level Programmiersprache, die just-in-time kompiliert wird. JavaScript ist besonders bekannt als Skriptsprache für Webseiten. Die Pakete in



JavaScript werden mittels dem Paketmanager npm verwaltet. (wird noch erweitert)

## 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. [2]

### 2.9.1. Redux Store

Redux ist ein State Container für JavaScript Apps, der es ermöglicht, States nicht mehr in den Komponenten, sondern in einem Zentralen Container zu speichern. Dadurch können beispielsweise States wiederhergestellt werden, nachdem eine Komponente geschlossen und wieder geöffnet wurde. Außerdem erleichtert Redux es, State Änderungen in Komponenten nachzuvollziehen. [4]

### 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.

## 3. Problemanalyse

An das MongoDB Visualisierungstool gibt es eine Reihe von Anforderungen, welche erfüllt werden müssen, damit das Projekt gelingen kann:

### 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. Deshalb muss das Frontend in einer gemeinsamen Webapp ausgeliefert werden (**FA1**). Diese Webapp muss in der Paketstruktur sowie in den verwendeten Komponenten modular sein (**FA2**). Um auf die einzelnen Anwendungen der Gesamtlösung zugreifen zu können, wird ein Startbildschirm benötigt, von welchem aus man die Applikationen starten kann (**FA3**).

Die Anwendung soll intuitiv benutzbar sein. Das bedeutet, dass ein Nutzer das System benutzen können muss, ohne vorher eine Benutzeranleitung oder ähnliches zu lesen (**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. Damit die Visualisierung großer Dokumente trotzdem einen Mehrwert hat, müssen diese unabhängig von der Größe übersichtlich und verständlich sein. Deshalb ist eine übersichtliche Darstellung der Schemata sehr wichtig (**FA6**).

Die Dokumente in einer Collection müssen nicht das gleiche Schema haben. Für einen Entwickler kann es hilfreich sein, einen Überblick über alle Variationen in einer Collection zu haben. Deshalb soll das MongoDB Visualisierungstool für jede Collection eine Detailansicht haben, die diese Variationen visualisiert. Der unterschied in diesen Variationen kann 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. Die Analyse sollte jedoch nicht länger als ein paar Sekunden dauern, da dies den Arbeitsfluss der Benutzer unterbrechen würde. Deshalb müssen die Datenbanken möglichst performant analysiert werden(**BA1**).

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

## 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** Möglichst performante Analyse

**BA2** Verbindung und Analyse mehrerer MongoDB Datenbanken gleichzeitig

# 4. Lösungskonzept

## 4.1. Bestehende Visualisierungstools

Es gibt bereits MongoDB Visualisierungstools auf dem Markt, jedoch erfüllt keines davon die zuvor definierten Anforderungen zu genüge:

### 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.

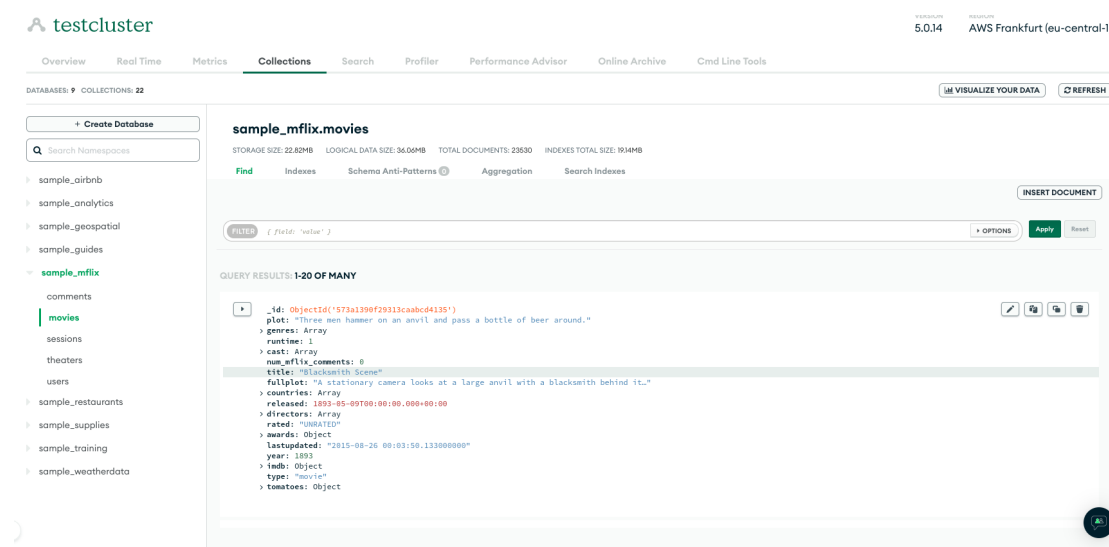


Abbildung 4.1.: MongoDB Data Explorer

### 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 an, 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.

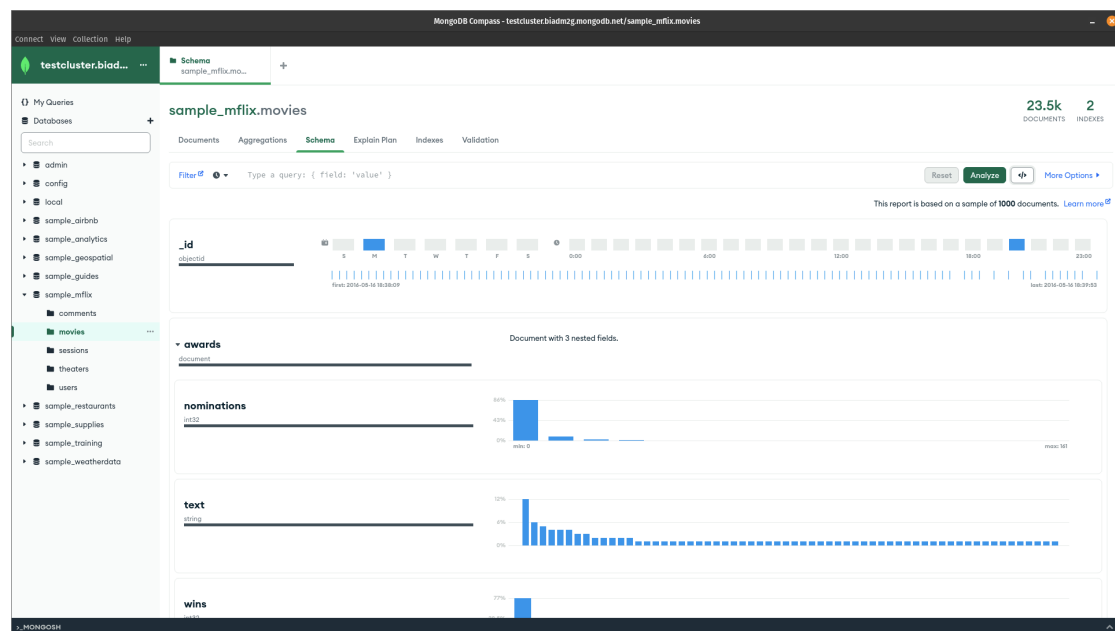


Abbildung 4.2.: MongoDB Compass

### 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.

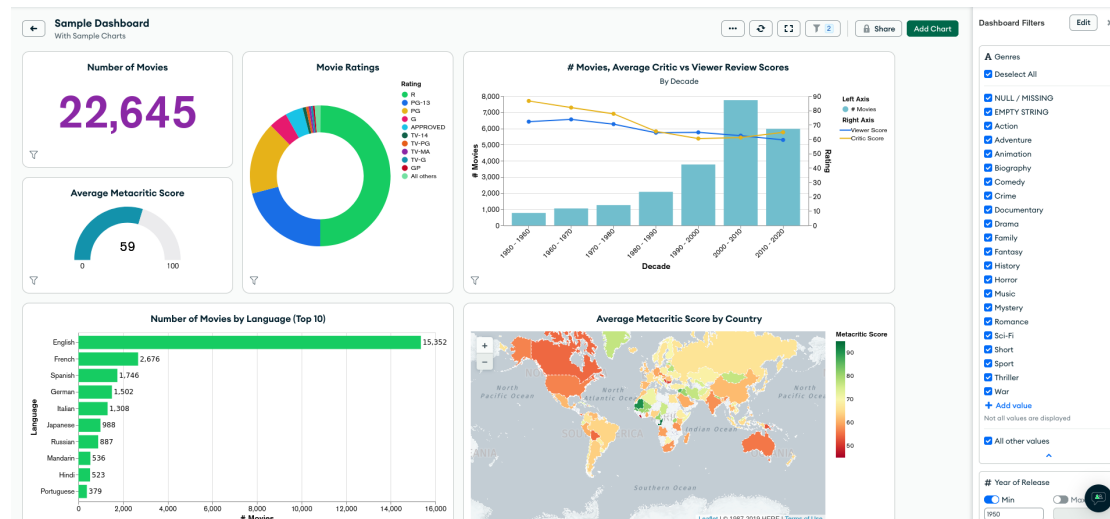


Abbildung 4.3.: MongoDB Charts

[1]

## 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-Framework 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 Spring Data MongoDB erfolgt jedoch mit festgelegten Datenbanken, welche in der application.properties Datei definiert werden. [13] Deshalb ist die Spring Data MongoDB Bibliothek für diese Anwendung nicht geeignet. Neben der Spring Data MongoDB Bibliothek gibt es auch noch einen anderen MongoDB 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. [7] 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. [14] 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 parallel 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. [10]

#### 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. [17] 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. [2] 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 Umsetzen von Anforderung **FA5** [9]

### 4.3. REST-Schnittstelle

Das Backend stellt 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. Wenn dies gelingt, wird die Datenbank ausgewertet und das Ergebnis als JSON im Response-Body zurückgegeben. Wenn die Verbindung fehlschlägt, wird der Statuscode 406 zurückgegeben.



## 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. Der Key entspricht dem Name des Feldes. Aus dem Value kann man den Datentyp auslesen. Ansonsten sind die Values nicht weiter von Relevanz, außer es handelt sich bei dem Datentyp um ein Array oder um einen Dictionary, also um ein verschachteltes Dokument. In diesen werden dann nämlich wieder Rekursiv die Key-Value Paare ausgelesen. Die Ergebnisse dieser Analyse werden in einer Objektorientierten Datenstruktur gespeichert: In der Klasse `ProcessedCollection` werden alle Variationen von Schemas gespeichert, inklusive dem Namen der Collection. Die Schemas selber werden in der Klasse `ProcessedDocument` gespeichert inklusive der Nummer an Dokumenten, die nach diesem Schema aufgebaut sind. `ProcessedDocument` enthält wiederum eine Liste an Values. Ein Value besteht aus einem Key, einem Typ, optional einer Referenz auf eine andere Collection (Wenn der Datentyp `ObjectId` ist und der Name nicht `id_` ist) und optional einem verschachtelten Dokument.

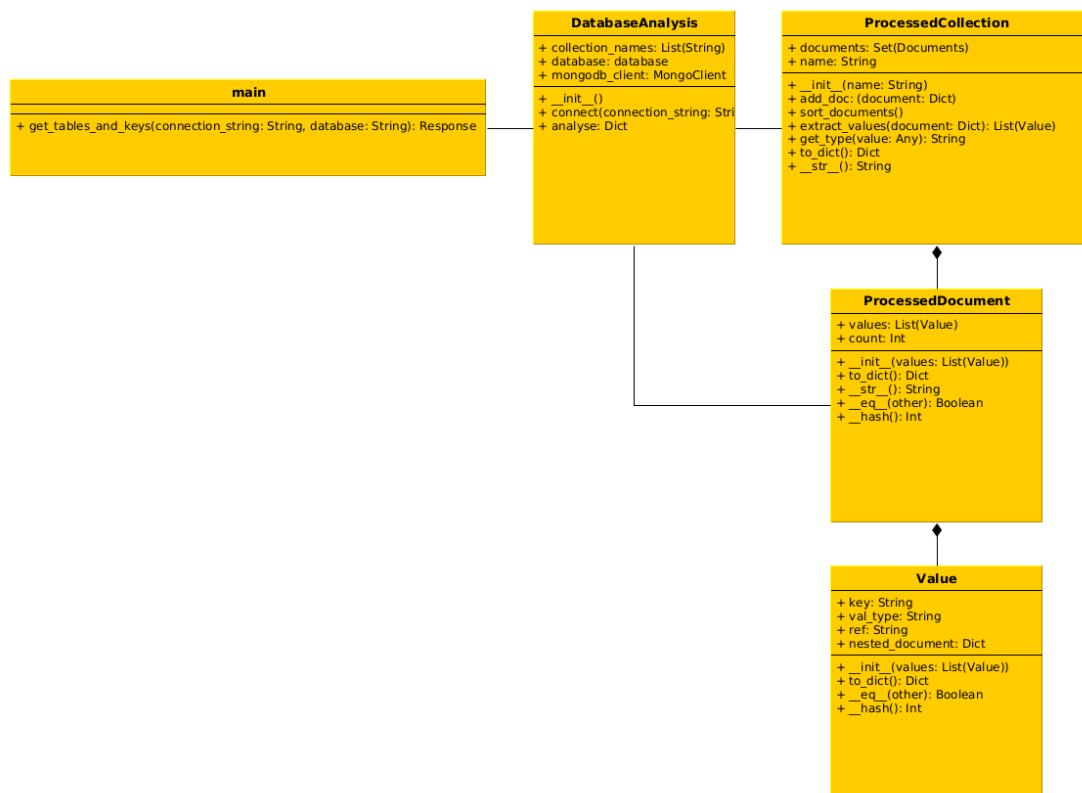


Abbildung 4.4.: Backend UML Diagramm

## 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.

Database connection		
Connection String		
Database		
<input type="checkbox"/> Option A		
<input type="checkbox"/> Option B		
Connect		
Feedback Label		

Person		
Key	Type	Reference
id_	Object	
name	String	
age	Number	
▼ cars	Array	

Car		
Key	Type	Reference
id_	Object ID	
model	String	
manufacturer	Object ID	Manufacturer
ps	Number	

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.

Database connection

Connection String

Database

☐ Option A

☐ Option B

Connect

Feedback Label

Person Detail View

This schema is used in 200 documents

Key	Type	Reference
id_	Object ID	
name	String	
age	Number	
cars	Array	

This schema is used in 130 documents

Key	Type	Reference
id_	Object ID	
name	String	
birth_date	Date	

Missing Values: Cars

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.

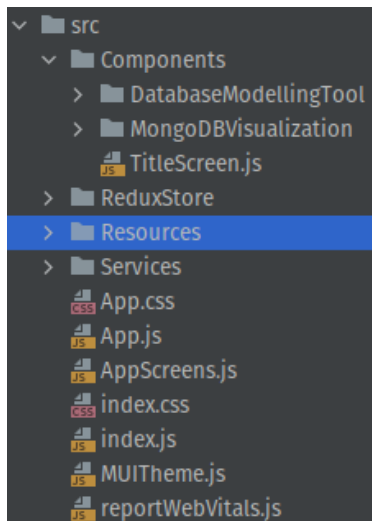


Abbildung 4.7.: Frontend Package Structure

Zudem wurde ein Titelbildschirm hinzugefügt, von 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 Titelbildschirm einen weiteren Button für die Anwendung hinzufügen.

## 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,
12                                         database_name, analyse_ref, sort_method)
13         connection_successful = db_analysis.connect()
14         if not connection_successful:
15             return Response(status=406)
16
17         document_dict = db_analysis.analyse()
18
19         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,
3                 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 Methode `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 =
2             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
11                 .post_processing(sort_method=self.sort_method)
12             if self.analyse_ref:
13                 self.analyse_references(processed_collection)
14             docs_dict["collections"]
14                 .append(processed_collection.to_dict())

```

---

```

15         self.mongoddb_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
8                     .append(new_doc.document_ages[0])
9                 doc.original_documents
10                .append(new_doc.original_documents[0])
11            return
12        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) /
11                                     len(document.document_ages)
12
13     def sort_documents(self, sort_method):
14         if sort_method == "documentCount":
15             self.documents = sorted(self.documents, key=lambda
16                                     document: document.count, reverse=True)
17         elif sort_method == "avgAge":
18             self.documents = sorted(self.documents, key=lambda
19                                     document: document.avg_age, reverse=True)
20
21     def mark_additional_values(self):
22         main_doc = self.documents[0]
23         for i in range(1, len(self.documents)):
24             for value in self.documents[i].values:
25                 if value not in main_doc.values:
26                     value.is_additional = True
27
28     def mark_missing_values(self):
29         main_doc = self.documents[0]
30         for i in range(1, len(self.documents)):
31             for value in main_doc.values:
32                 if value not in self.documents[i].values:
33                     self.documents[i]
34                     .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 =
8                 document.get("_id").generation_time
9             document_age =
10                 (datetime.datetime.now(datetime.timezone.utc) -
11                  document_generation_time).total_seconds()
12             self.document_ages.append(document_age)
13         self.avg_age = None
14         self.missing_values = list()
15         self.original_documents = list()
16         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 simples 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.

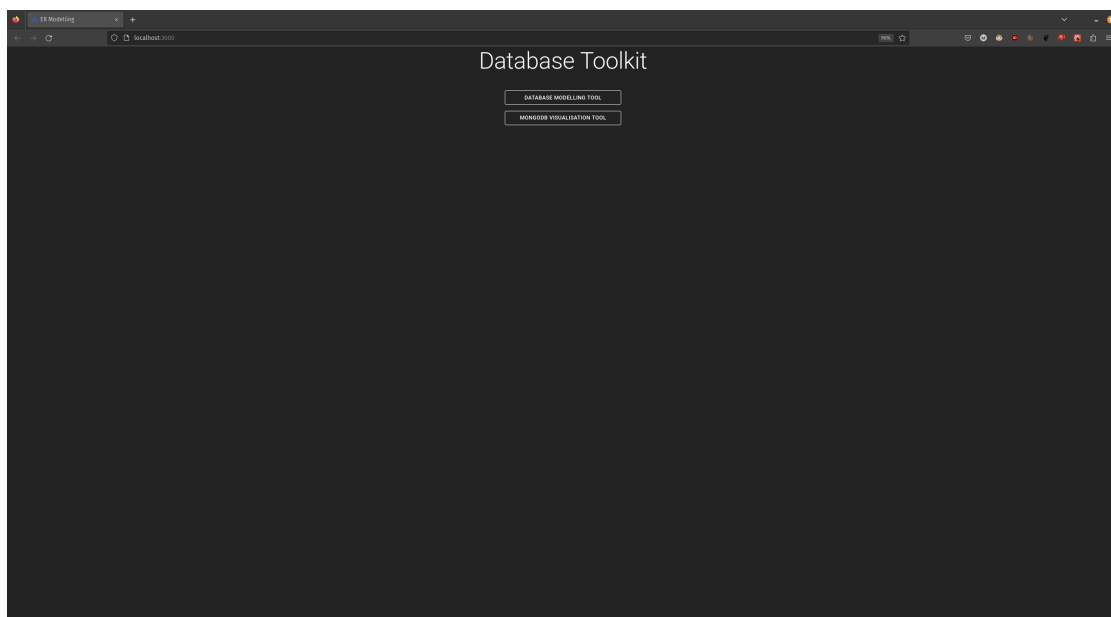


Abbildung 5.1.: Frontend Startbildschirm

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}
5                  element={<DatabaseModellingTool/>}/>
6              <Route path={AppScreens.mongoVisTool}
7                  element={<MongoManager/>}/>
9          </Routes>
10     </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)

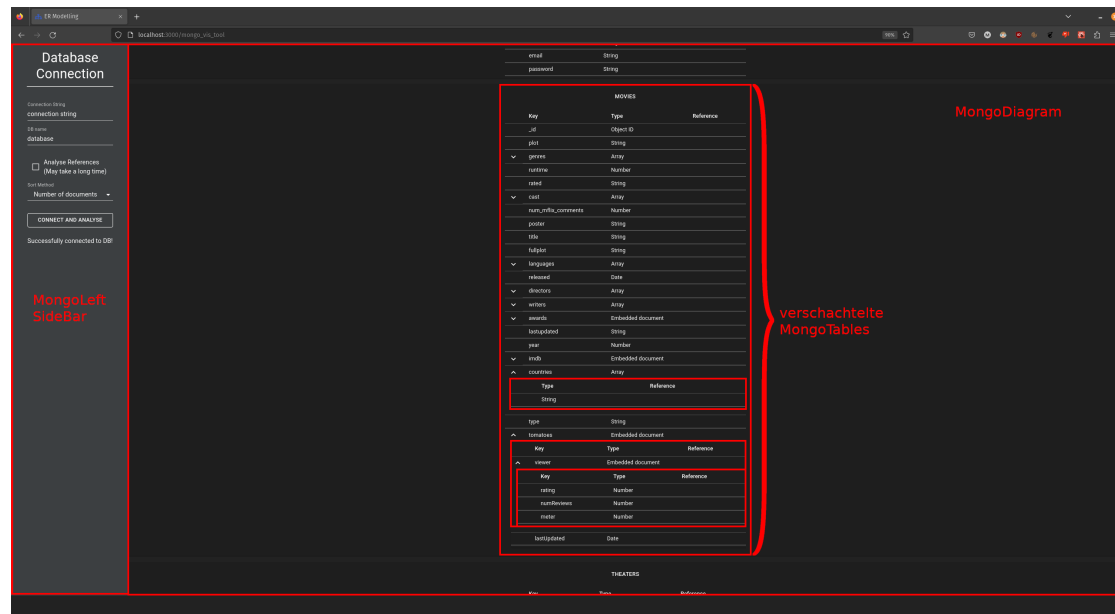


Abbildung 5.2.: Frontend Elemente

### 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 in 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 verschachtelte Untertabelle, entspricht also dem Datentyp Embedded Document.
- **array** wird für den Datentyp Array benutzt.
- **detail** wird im Detail View Popup benutzt, um alle Variationen 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.

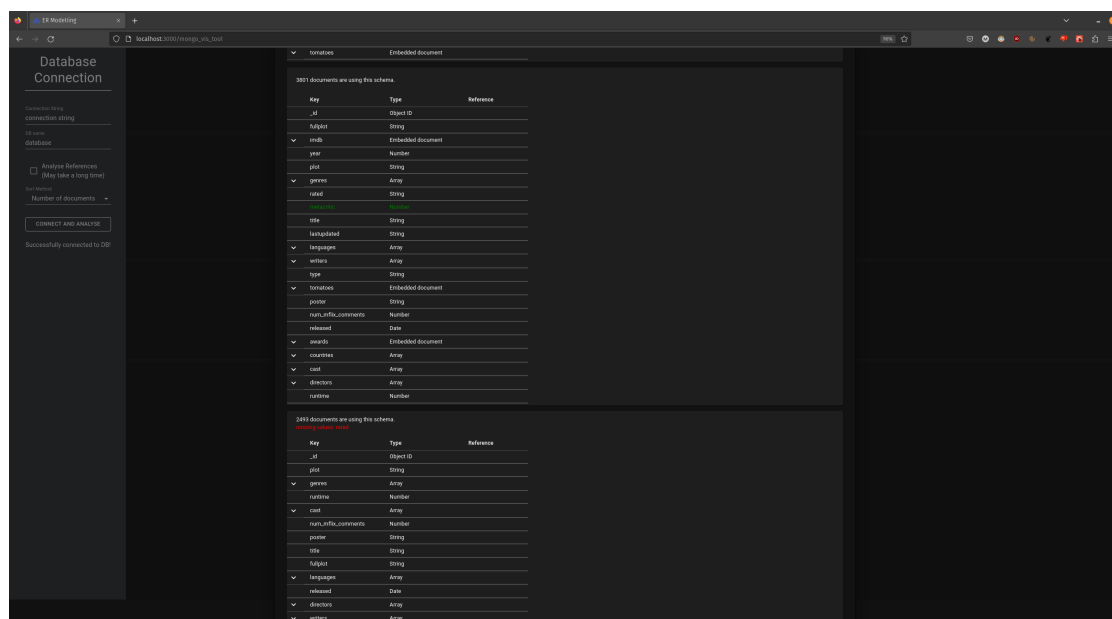


Abbildung 5.3.: Frontend Detail View Popup

### 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.

---

```
1   export const mongoContentSlice = createSlice({
2     name: 'mongoContent',
3     initialState: {
4       collections: null
5     },
6     reducers: {
7       setCollections: (state, documents) => {
8         state.collections = documents
9       }
10    }
11  })
12
13  export const {setCollections} = mongoContentSlice.actions
14  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.

---

```
1   export const store = configureStore({
2     reducer: {
3       erContent: erContentSlice,
4       relationalContent: relationalContentSlice,
5       mongoContent: mongoContentSlice
6     }
7   })
8
9   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 ist uWSGI und Nginx vorinstalliert, was das ausliefern von Flask Applikationen erleichtert. [15]

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

Das Frontend wurde mit 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.

## 7. Evaluierung

Aufgabe des Kapitels Evaluierung ist es, in wie weit die Ziele der Arbeit erreicht wurden. Es sollen also die erreichten Arbeitsergebnisse mit den Zielen verglichen werden. Ergebnis der Evaluierung kann auch sein, dass bestimmte Ziele nicht erreicht werden konnten, wobei die Ursachen hierfür auch außerhalb des Verantwortungsbereichs des Praktikanten liegen können.

## **8. Zusammenfassung und Ausblick**

### **8.1. Erreichte Ergebnisse**

Die Zusammenfassung dient dazu, die wesentlichen Ergebnisse des Praktikums und vor allem die entwickelte Problemlösung und den erreichten Fortschritt darzustellen. (Sie haben Ihr Ziel erreicht und dies nachgewiesen).

### **8.2. Ausblick**

Im Ausblick werden Ideen für die Weiterentwicklung der erstellten Lösung aufgezeigt. Der Ausblick sollte daher zeigen, dass die Ergebnisse der Arbeit nicht nur für die in der Arbeit identifizierten Problemstellungen verwendbar sind, sondern darüber hinaus erweitert sowie auf andere Probleme übertragen werden können.

#### **8.2.1. Erweiterbarkeit der Ergebnisse**

Hier kann man was über die Erweiterbarkeit der Ergebnisse sagen.

#### **8.2.2. Übertragbarkeit der Ergebnisse**

Und hier etwas über deren Übertragbarkeit.

# 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] Alex Banks und Eve Porcello. *Learning React: Modern Patterns for Developing React Apps*. O'Reilly Media, 2020.
- [3] Shannon Bradshaw, Eoin Brazil und Kristina Chodorow. *MongoDB: the definitive guide: powerful and scalable data storage*. O'Reilly Media, 2019.
- [4] 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).
- [5] Miguel Grinberg. *Flask web development: developing web applications with python*. O'Reilly Media, Inc.", 2018.
- [6] *Introducing JSON*. URL: <https://www.json.org/json-en.html> (besucht am 11.08.2022).
- [7] Selina Khoirom u. a. „Comparative analysis of Python and Java for beginners“. In: *Int. Res. J. Eng. Technol* 7.8 (2020), S. 4384–4407.
- [8] *MongoDB Systemeigenschaften*. 2023. URL: <https://db-engines.com/de/system/MongoDB> (besucht am 30.01.2023).
- [9] *MUI: The React component library you always wanted*. 2023. URL: <https://mui.com/> (besucht am 30.01.2023).
- [10] *PyMongo 4.3.3 Documentation*. 2022. URL: <https://github.com/mongodb/mongo-python-driver> (besucht am 30.01.2023).
- [11] Edwin Schicker. *Datenbanken und SQL*. 5. Auflage. Regensburg: Springer, 2017. ISBN: 978-3-658-16128-6.
- [12] 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.
- [13] *Spring Data MongoDB*. 2023. URL: <https://spring.io/projects/spring-data-mongodb> (besucht am 30.01.2023).
- [14] Söderlund Sverker. „Performance of REST applications“. Diss. 2017.
- [15] *uwsgi-nginx-flask*. 2023. URL: <https://hub.docker.com/r/tiangolo/uwsgi-nginx-flask> (besucht am 21.02.2023).

- [16] Guido Van Rossum und Fred L Drake Jr. *Python tutorial*. Bd. 620. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 2020.
- [17] 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**