

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

Eidesstattliche Erklärung	i
Kurzfassung	ii
Inhaltsverzeichnis	iii
Abbildungsverzeichnis	vi
Quelltextverzeichnis	vii
Abkürzungsverzeichnis	viii
1. Einleitung	1
1.1. Motivation	1
1.2. Problemstellung und -abgrenzung	1
1.3. Vorangegangene Arbeit	2
1.4. Ziel der Arbeit	2
2. Grundlagen	3
2.1. JSON	3
2.2. HTTP	5
2.3. SQL	5
2.3.1. Normalformen	5
2.4. MongoDB	5

2.5. REST API	6
2.6. Python	6
2.7. Flask	6
2.8. JavaScript	7
2.9. React	7
2.9.1. Redux Store	7
2.9.2. Axios	7
3. Problemanalyse	8
4. Lösungskonzept	9
4.1. Bestehende Visualisierungstools	9
4.1.1. MongoDB Data Explorer	9
4.1.2. MongoDB Compass	10
4.1.3. MongoDB Charts, Tableau, Qlik und Looker	10
4.2. Verwendete Technologien	11
4.2.1. Backend Technologien	11
4.2.2. Frontend Technologien	12
4.3. REST-Schnittstelle	13
4.4. Analyse der MongoDB Datenbank	13
4.5. Planung des Frontends	14
4.6. Modularität des Frontends	14
5. Implementierung	15
5.1. Backend	15
5.2. Frontend	18
6. Inbetriebnahme	19

7. Evaluierung	20
8. Zusammenfassung und Ausblick	21
8.1. Erreichte Ergebnisse	21
8.2. Ausblick	21
8.2.1. Erweiterbarkeit der Ergebnisse	21
8.2.2. Übertragbarkeit der Ergebnisse	21
Literatur	22
A. Anhang A	24
B. Anhang B	25

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	9
4.2. MongoDB Compass	10
4.3. MongoDB Charts	11
4.4. Backend UML Diagramm	14

Listings

5.1. app.py	15
5.2. DatabaseAnalysis.connect	16
5.3. DatabaseAnalysis.analyse	16
5.4. DatabaseAnalysis.analyse_references	17
5.5. ProcessedCollection.add_doc	17

Abkürzungsverzeichnis

RUP	Rational Unified Process	19
SQL	Structured Query Language	5
HTTP	Hypertext Transfer Protocol	5
API	Application Programming Interface	6
REST	Representational State Transfer	6
JSON	JavaScript Object Notation	3
UI	User Interface	7
DDL	Data Definition Language	5
DML	Data Manipulation Language	5
XML	Extensible Markup Language	6
JAX-RS	Jakarta RESTful Web Services	6
POJO	Plain Old Java Object	11

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 Nutzeroberfläche, 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. [6]

Object ist ein Set aus Key/Value Paaren.

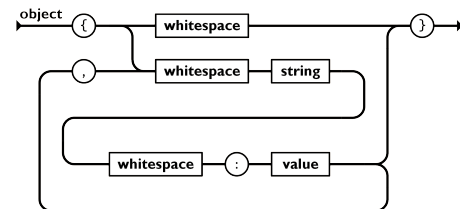


Abbildung 2.1.: JSON Object

Array ist eine geordnete Liste von Values.

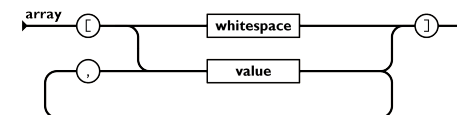


Abbildung 2.2.: JSON Array

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

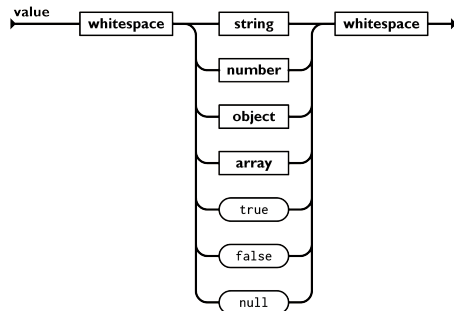


Abbildung 2.3.: JSON Value

String ist eine Sequenz aus Unicode Buchstaben.

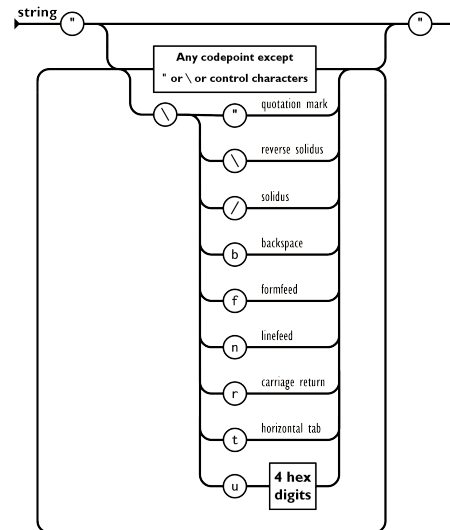


Abbildung 2.4.: JSON String

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

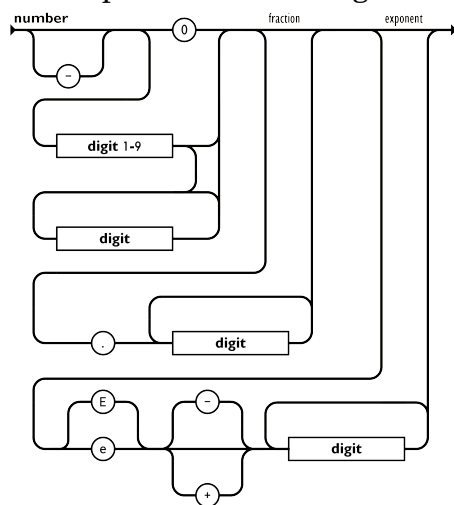


Abbildung 2.5.: JSON Number

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. 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]

2.3.1. Normalformen

2.4. 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, dass 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. [\[3\]](#)

2.5. 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. [\[12\]](#)

2.6. 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. [\[15\]](#)

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.

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 JacaScript 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:

- A1** Das Visualisierungstool muss das bestehende Er Modellierungstool erweitern.
- A2** Das Gesamtsystem muss weiterhin modular sein, sodass das System ohne Änderungen an den bestehenden Komponenten um weitere Komponenten erweitert werden kann.
- A3** Die Analyse der MongoDB muss performant genug sein, um Datenbanken in wenigen Sekunden analysieren zu können.
- A4** Das Frontend muss beliebig aufgebaute MongoDB Dokumente anschaulich visualisieren können.
- A5** Die Anwendung muss von mehreren Nutzern gleichzeitig benutzbar sein.

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.

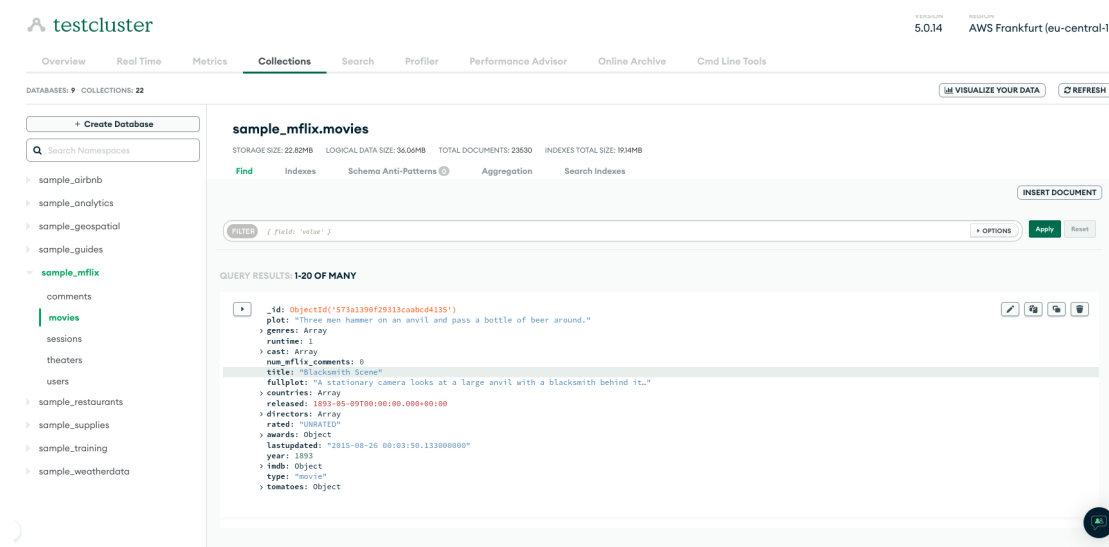


Abbildung 4.1.: MongoDB Data Explorer

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.

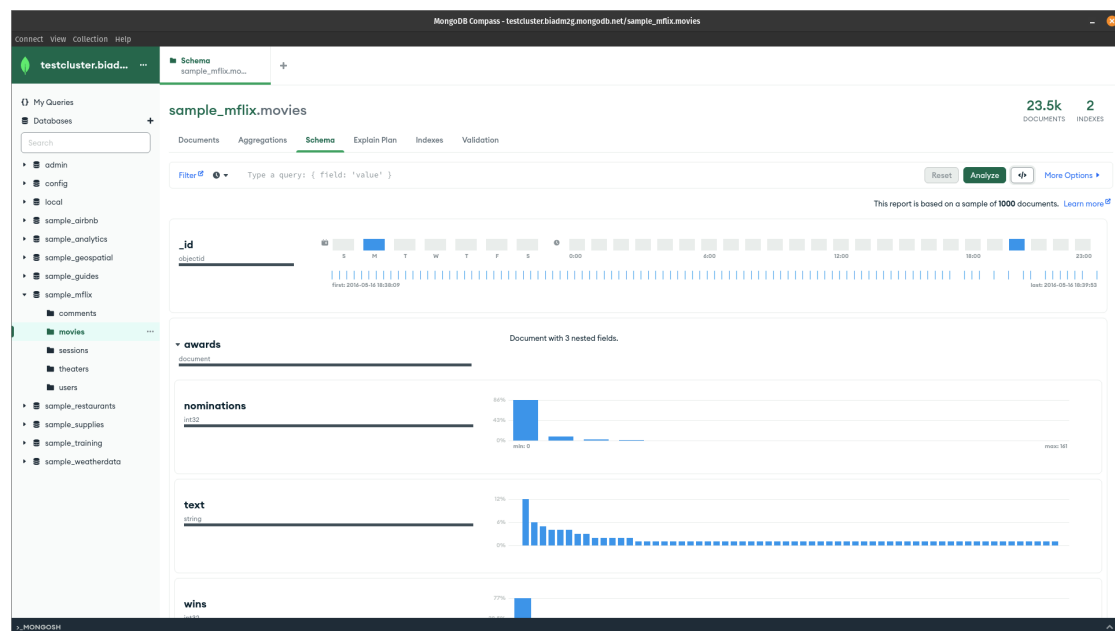


Abbildung 4.2.: MongoDB Compass

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.

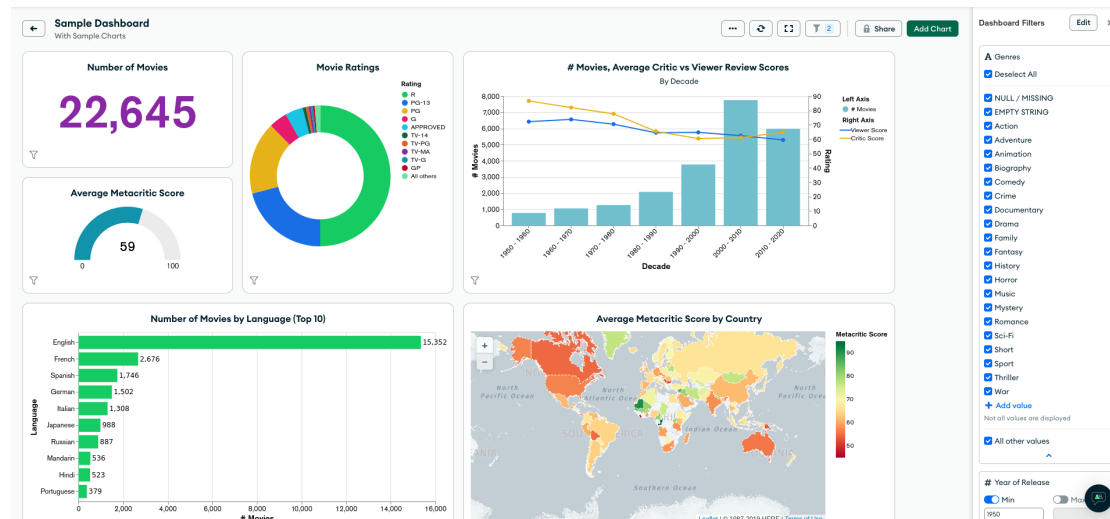


Abbildung 4.3.: MongoDB Charts

[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 A5 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 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. [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 **A3**. 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 **A3** 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. [16] 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 **A1** erfüllt werden kann. Da React dank React Elements und React Components in seiner Grundstruktur sehr modular ist, eignet sich React sehr gut, um Anforderung **A2** 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 an vielen das Definieren von 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. [9]

4.3. REST-Schnittstelle

Das Backend stellt einen einzigen Endpunkt bereit: `/connect` erwartet im Body des Requests einen String `connection_string` und einen String `database` im JSON-Format. 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 Py-Mongo 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 einen Dictionary, also um ein verschachteltes Dokument. In diesem 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` ent-

hält 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.

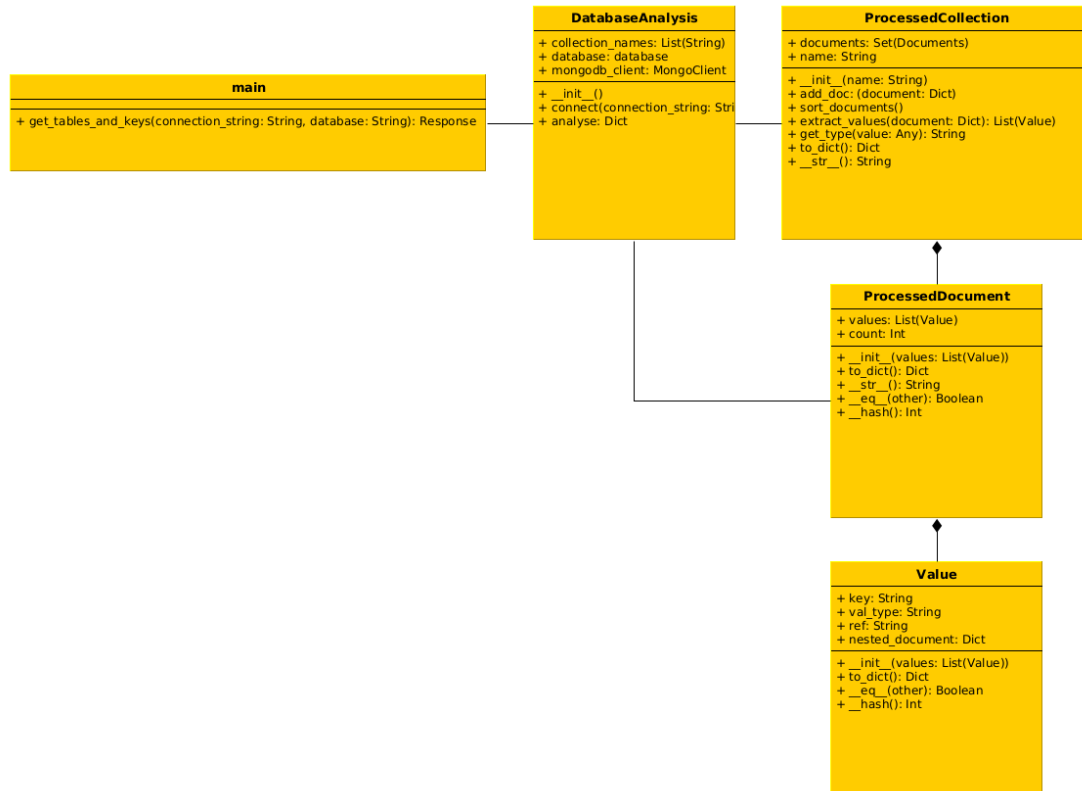


Abbildung 4.4.: Backend UML Diagramm

4.5. Planung des Frontends

4.6. Modularität des Frontends

5. Implementierung

5.1. Backend

Der bereits beschriebene 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         return json.dumps(document_dict)
```

Quelltext 5.1: app.py

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,
4                                                  serverSelectionTimeoutMS=5000)
5             self.database = self.mongodb_client[database]
6         except pymongo.errors.ServerSelectionTimeoutError:
7             return False
8         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. 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 =
3             self.database.list_collection_names()
4         docs_dict = {"collections": []}
5         for name in self.collection_names:
6             processed_collection = ProcessedCollection(name)
7             collection = self.database.get_collection(name)
8             documents = collection.find({})
9             for document in documents:
10                 processed_collection.add_doc(document)
11
12             processed_collection
13                 .post_processing(sort_method=self.sort_method)
14             if self.analyse_ref:
15                 self.analyse_references(processed_collection)
16             docs_dict["collections"]
17                 .append(processed_collection.to_dict())
18         self.mongodb_client.close()
19         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 Object ID ist und der Name des Values nicht _id ist (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 Referent 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 wurde, wird die 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 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

5.2. Frontend

5.2.1.

6. Inbetriebnahme

Aufgabe des Kapitels Inbetriebnahme ist es, die Überführung der in Kapitel 5 entwickelte Lösung in das betriebliche Umfeld aufzuzeigen. Dabei wird beispielsweise die Inbetriebnahme eines Programms beschrieben oder die Integration eines erstellten Programmodules dargestellt.

Bei der Software-Erstellung entspricht dieses Kapitel der Auslieferungsphase (Deployment) im Rational Unified Process (RUP).

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

- [16] 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