Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Verteilte Systeme
Data Science und Digital Libraries
Studiengang Informatik (B.Sc.)

Nachschlagedienst für qualitative hochwertige Metadaten von Veröffentlichungen

Bachelorarbeit

vorgelegt von
Nam Dan Nhu (10011937)
am 08.06.2022

Betreuer:	Dr. rer. nat. Oliver Karras
Erstprüfer:	Prof. Dr. Sören Auer
Zweitprüfer:	Dr. rer. nat. Oliver Karras

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden, alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind, und die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen hat.

Hannover, den 08.06.2022

Nam Dan Nhu

Zusammenfassung

Metadaten, die auch als "Daten von Daten" bezeichnet werden, werden in Dienste wie der *Open-Research-Knowledge-Graph* (ORKG), die darauf angewiesen sind, genutzt. Dabei sind qualitativ hochwertige Metadaten (von Publikationen) erforderlich, um das Potential solcher Dienste ausschöpfen zu können. Es gibt verschiedene Dienstanbieter für Literaturrecherche, die Metadaten zu Publikationen liefern können. Jedoch variiert die Qualität der Metadaten von Dienstanbieter zu Dienstanbieter. Außerdem wird die Qualität der Daten häufig als "fitness for pupose" verstanden.
In dieser These verfolgt man das Ziel: Die Lieferung qualitativ hochwertiger Metadaten von Publikation über eine Schnittstelle, um solche Dienste wie der ORKG bei der Versorgung von hochwertigen Metadaten zu unterstützen.
Um dieses Ziel zu erreichen, wird eine REST-API mit dem Web-Framework *FastAPI* gebaut, die als Schnittstelle dient, um Metadaten aus verschiedenen Dienstanbietern zu vergleichen und den Datensatz mit den hochwertigsten Metadaten zurückzuliefern. Es werden dafür Qualitätsaspekte (Dimensionen) betrachtet und priorisiert. Die Metriken der 5 Qualitäts-Dimensionen (Availability, Completeness, Accuracy, Understandability und Objecticity) mit der höchsten Prioritätsstufe werden definiert und für die Schnittstelle implementiert.
Messungen und Tests wurden durchgeführt, um die Geschwindigkeit und Funktionalität der Schnittstelle zu ermitteln. Die Ergebnisse der Messungen und Tests zeigen eine hohe Ähnlichkeit des Qualitätsgrades bei den gelieferten Metadaten und denen der ORKG, und die Abfrage über den Publikationstitel führt zu einer erhöhten Berechnungszeit von 40% im Vergleich zu einer Abfrage über die DOI.
Es konnte dadurch geschlossen werden, dass die Qualität der aus der Rest-API gelieferten Metadaten für den ORKG-Dienst geeignet ist und die Nutzung der Schnittstelle mit einer DOI priorisiert werden soll, wenn eine höhere Anzahl an Metadaten-Abfrage gewünscht ist.

Abstract

Metadata, also defined as "data of data", is used in services like the "Open-Research-Knowledge-Graph" (ORKG). Yet, metadata of publications with a high-quality degree is necessary to maximize the potential of such services. There are plenty of service providers for literature research, that manage and provide metadata of publications. However, the metadata quality varies from provider to provider. Quality of data is commonly conceived as "fitness for purpose".
This thesis aims to develop a REST-API, that serves the purpose to deliver high-quality metadata to the ORKG-service. In order to achieve this goal, a REST API was built with the help of the web framework *FastAPI*, so it can compare metadata of publications from multiple service providers for literature research to yield a dataset consisting of metadata with the highest quality degree. Quality aspects (also called dimensions in this thesis) were considered and prioritized to determine a quality degree of metadata. For each of the 5 quality dimensions (Availability, Completeness, Accuracy, Understandability und Objecticity) with the highest priority level, a respective metric was defined and implemented.
To validate the applicability of the developed REST-API, measurements, and tests were executed. Results of the measurements and tests indicate high similarity between the yielded metadata and these from the ORKG. Results related to the performance speed show a 40% reduction in this area when querying metadata with the use of a publication title instead of a DOI.
These results conclude, that the measured quality of metadata the developed REST API yields seems to fit in the ORKG-service and the user should prioritize querying metadata with a DOI instead of a publication title if a high query number is wished.

Inhaltsverzeichnis

1 Einleitung

Verschiedene Dienste, die auf Metadaten wissenschaftlicher Publikationen angewiesen sind, haben eine große Auswahl an Quellen, von denen man die Metadaten abrufen kann. Der Abruf der Metadaten sowie das Format, indem die Metadaten geliefert werden, sind über die Quellen hinweg nicht einheitlich. Ein weiteres Problem ist, dass nicht jede Quelle Metadaten zu jeder wissenschaftlichen Publikation enthält. Zudem haben die Metadaten in den gelieferten Datensätzen der verschiedenen Quellen unterschiedliche Qualitäts-Grade. Dazu unterscheiden sich die Datensätze in der Vollständigkeit, Darstellung und den genutzten Vokabularen.
Um das Potential solcher Dienste ausnutzen zu können, ist es entscheidend Datensätze mit qualitativ hochwertige Metadaten zu liefern. Jedoch ist die Definition der Qualität nicht eindeutig und muss somit noch definiert werden. Im Fokus der These steht der Dienst "*Open Research Knowledge Graph*" (ORKG) von der Tib.

1.1 Definition der Ziele

Für die These existieren mehrere Ziele, die verfolgt werden. Das Hauptziel der These ist die Entwicklung einer Schnittstelle, mit der qualitativ hochwertige Metadaten von Publikationen abgefragt werden können. Um das zu erreichen müssen verschiedene Dienstanbieter für Literaturrecherchen analysiert und miteinander verglichen werden, um zu ermitteln, welche Dienstanbieter als Quellen der Metadaten Abrufe dienen werden. Ein essentielles Ziel für die Qualitätsbewertung der Metadaten, ist die Definition der Qualität durch die

Analyse, Definition, Priorisierung und Auswahl der Qualitätsaspekte (-dimensionen). Für die Messungen der Grade von ausgewählten Qualitätsdimensionen, müssen definiert und implementiert werden. Außerdem müssen User-Stories und Use-Cases definiert und implementiert werden, um die Bedürfnisse eines Nutzers bei der Anwendung der gebauten REST-API zu erfüllen. Dabei ist es wünschenswert, wenn die gebaute REST-API eine Abfrage innerhalb von 3 Sekunden bearbeiten kann.

1.2 Struktur der These

Die Struktur der These ist folgendermaßen aufgebaut. Das erste Kapitel dient als eine Einführung in die Thematik, worin erläutert wird, was die Motivation hinter der These ist, welche Lösung vorgeschlagen wird und wie die These aufgebaut ist. Im zweiten Kapitel werden die Grundlagen erläutert, sodass man bestimmte Grundkenntnisse erhält, die man für die darauffolgenden Kapitel benötigt. Das dritte Kapitel beschäftigt sich mit verwandten Arbeiten, worauf die These basiert. In Kapitel 4 werden User-Stories und Use-Case-Tabellen definiert, die einen Überblick über die spätere Implementierung geben. Kapitel 5 analysiert die Dienstanbieter, die die Quellen der Metadaten darstellen. In dem darauffolgenden Kapitel 6 werden die User-Stories sowie alle Metriken implementiert. Das 7.Kapitel befasst sich mit der Evaluation des implementierten Programms, um die Funktionalität des Programms zu bewerten sowie der Vergleich der einzelnen Werte und der Zweck den sie repräsentieren. Das letzte Kapitel dient zur Beurteilung der Arbeit und fügt einige Ideen für zukünftige Arbeiten hinzu, um die Schnittstelle und Qualitätsbewertung zu verbessern.

2 Grundlagen

2.1 Metadaten

Metadaten sind ein essentieller Bestandteil dieser Arbeit und wird aus dem Grund in diesem Kapitel behandelt. Hier wird die Definition, Charakteristiken und Arten von Metadaten abgedeckt, sodass man ein grundlegendes Verständnis und ein Bild davon erhält.
Metadaten haben verschiedene Bedeutungen - eines davon ist "Daten von Daten" [1, 2, 3, 4], welches ohne weitere Interpretation recht unklar ist. Somit wird indirekt gedeutet, dass man zuerst verstehen muss, was "Daten" sind. Dafür werden 2 Zeilen aus dem Gedicht "The Rock"von T.S. Eliot zitiert [3]:

> Where is the wisdom we lost in knowledge?
> Where is the knowledge we lost in information?

Eine Analyse der beiden Zeilen führt zu dem Rückschluss, dass Eliot eine Hirarchie in fallender Reihenfolge aufstellt, in der an oberster Stelle die Weisheit (wisdom) ist, gefolgt von Wissen (knowledge) und an unterster Stelle die Information (information). In dieser Hirarchie fehlt noch ein Begriff, welches der Information untergeordnet ist. Um es zu vervollständigen, wird der Begriff "Daten" angehangen. Die abgeleitete Hirarchie, bestehend aus (1) Weisheit, (2) Wissen, (3) Information und (4) Daten, beschreibt das informative Level.
Die beiden Abbildungen [2.1, 2.2] geben ein Beispiel zu den Metadaten einer Pulikation und es wird mit dessen Hilfe die Hirarchie erläutert:

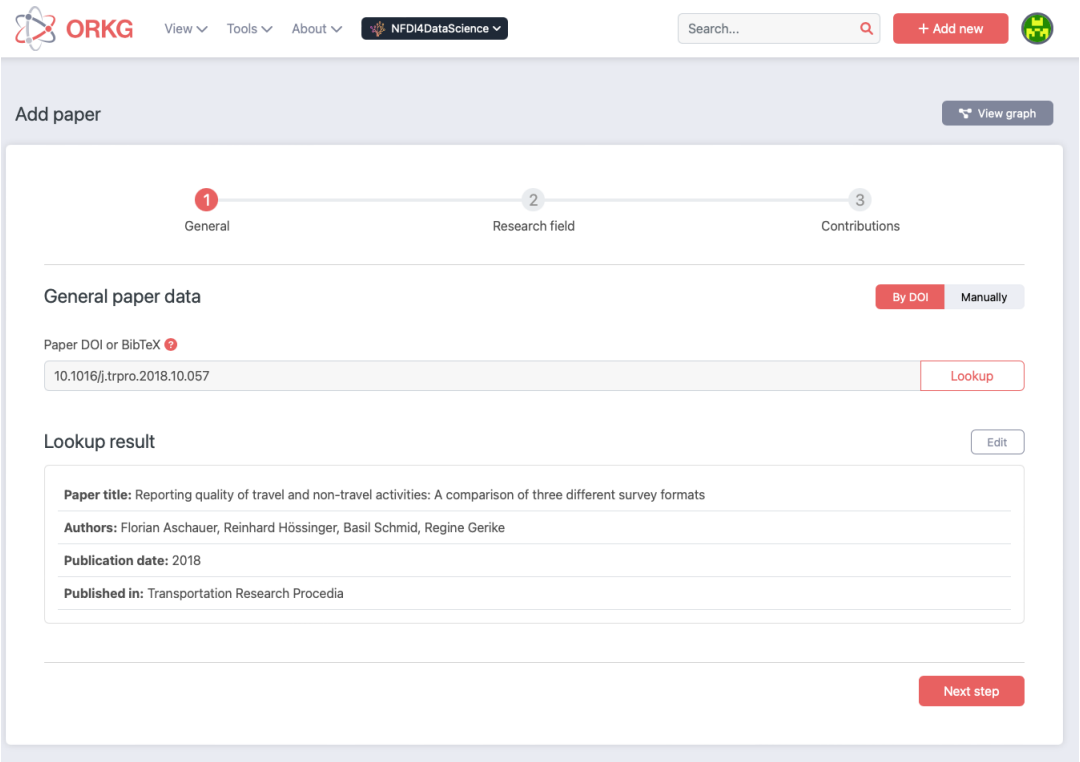

Abbildung 2.1: ORKG-Service

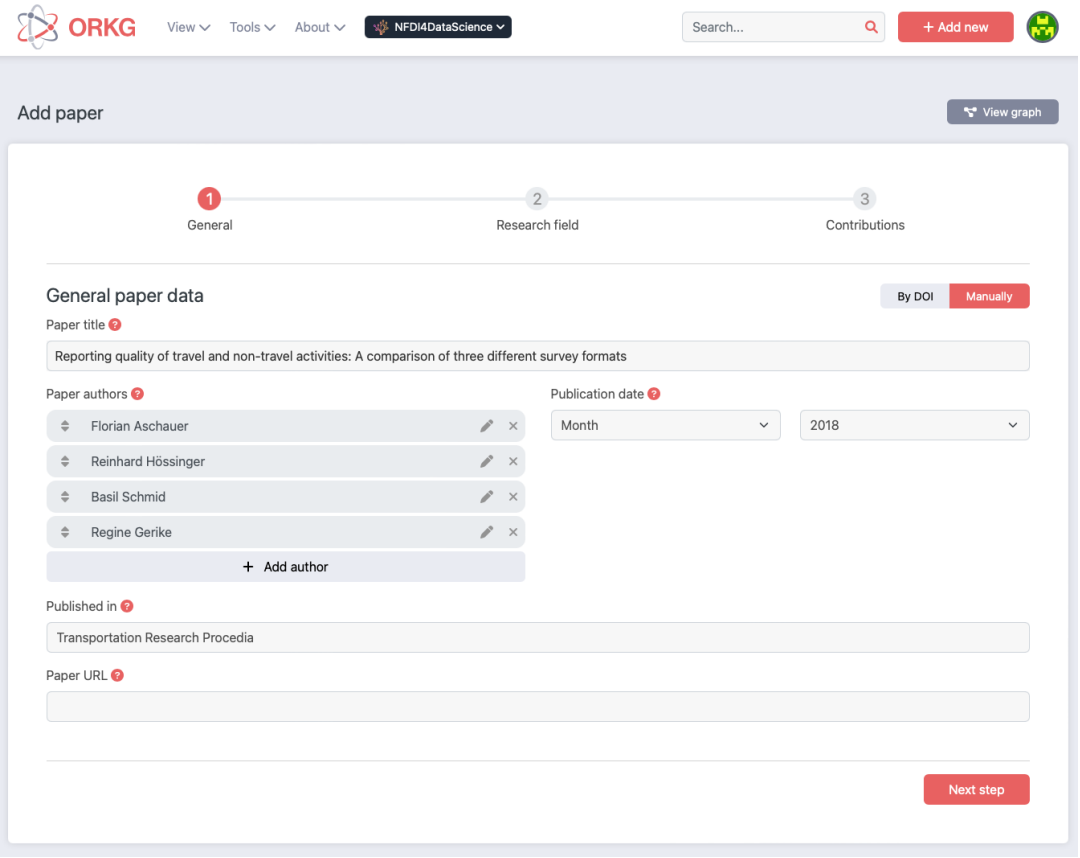

Abbildung 2.2: ORKG-Service

Werte, die man aus den 2 Abbildungen etziehen kann sind:
10.1016/j.trpro.2018.10.057; *Reporting quality of travel and non-travel activities: A comparison of three different survey formats*; *Florian Aschauer*, *Reinhard Hössinger*; *Basil Schmid*; *Regine Gerike*; *2018* und *Transportation Research Procedia*.

Daten: Die einzelnen Werte folgen eine bestimmte Syntax, ergeben ohne einen Kontext keinen Sinn und man kann nicht eindeutig erkennen, was sie darstellen sollen [5, 6].
Information: Nachdem man Daten eine Bedeutung gibt, können Daten in einem gegebenen Kontext interpretiert werden [5, 6]. Man stellt sich die Frage: Wofür stehen diese Daten?
Wissen: Entsteht durch die Verknüpfung der Informationen [6], d.h. Zusammenhänge zwischen Informationen erkennen und diese miteinander verknüpfen (z.B. Titel der Publikation gibt Informationen über das Forschungsgebiet aus.).
Weisheit: Anwendung/Umsetzung des Wissens (z.B. (1)Abstract analysieren und beurteilen, ob es für die weitere Forschung nützlich ist oder (2) Kombination von Schlüsselwörter aus dem Titel bilden, um ähnliche Publikationen aus dem Forschungsgebiet zu finden).

2.1.1 Definition

Es wird jetzt nochmal der Ausschnitt "Daten von Daten" aufgegriffen, das Metadaten definieren soll. Gartner [2] und Pomeratz [3] haben in deren Arbeiten beschrieben, dass der Ursprung der Metadaten von der Katalogisierung aus der Bibliothek stammt und dessen Zweck ist: bei der Suche aus der Bibliothekssammlung zu unterstützen. Dabei werden Metadaten von Gartner [2], Pomeratz [3] und Baca [1] beschrieben als ein Informationsobjekt, das ein Informationsobjekt beschreibt.
Baca [1] motiviert in seiner Arbeit, dass man Metadaten als eine Summe von Dingen betrachten soll, die man zu einem Informationsobjekt sagen kann. Wenn man den Gedankengang folgt, führt es zu der Beschreibung [7]:

Informationsobjekte tragen bestimmte Metadaten in sich, die unter den Umständen der Erstellung, Verwaltung und Nutzen ganz natürlich entstehen. Haynes zitiert Qins und Zengs [8] Definition des Begriffs *Metadaten* " Metadaten fassen Informationen eines informationstragenden Entität zusammen". In Haynes Buch [4] gibt er ein Beispiel, worin Qin und Zeng über Lebensmitteletiketten als Metadaten reden. Es wurde darin ausgedrückt, dass die Daten eines Objektes, das keine Informationsressource ist, keine Metadaten und nichts weiter als Daten sind. Rühle [9] fasst all diese Definitionen zusammen: Metadaten sind strukturierte Daten, die für die einheitliche Beschreibung jeglicher Art von Ressourcen dienen. Solche Ressourcen-Arten wären z.B. Gemälde, Gebäude, Orte, Publikationen, usw. ... Deren Zweck ist es eine einheitliche Struktur der Beschreibung für Instanzen von ein- und derselben Ressourcenklasse zu geben, um die Suche, das Finden und das Selektieren relevanter Ressourcen aus einer Menge von Ressourcen zu erleichtern.

2.1.2 Kategorien/Arten von Metadaten

Strukturelle Metadaten

Der Name dieser Metadaten-Kategorie, weist auf den Zweck, den es erfüllen soll, hin. Solche Metadaten beschreiben die Struktur archivierter Objekte. Der Hintergedanke hierbei ist die Erhaltung der ursprünglichen Darstellungsform für den Nutzer. Dabei berücksichtigt die interne Struktur physische als auch logische Beziehungen zwischen den einzelnen Komponenten (wie ein Buch, das in Seiten gegliedert ist, oder eine Website, die aus vielen verschiedenen Dateien besteht) [7].

Administrative Metadaten

Administrative Metadaten befasst sich mit den Informationen, die den Archivierungsprozess, den zukünftigen Zugriff und die authentische Wiedergabe archivierter Ressourcen beschreibt [7].

Deskriptive Metadaten

Heine [7] hebt bei der Definition dieser Metadaten-Kategorie hervor, dass der Name "*deskriptiv*" ungünstig gewählt wurde, weil im Prinzip alle Metadaten eine Ressource beschreiben. Jedoch bezieht sich diese Kategorie speziell auf die bibliografischen Informationen (wie Author, Titel, usw.), die aus der Nutzersicht für das Auffinden einer Ressource erforderlich sind.

2.2 REST-API

2.2.1 Definition

Um das Verständnis des Begriffes "*REST-API*" zu verstehen, wird es in 2 Teile aufgeteilt: REST und API.
Eine API (Application Programming Interface) ist eine Schnittstelle, die Funktionen und Daten freigibt, um die Interaktion zwischen Programmen zu erleichtern, sodass sie Informationen austauschen können [10]. REST ist die Abkürzung für "*Representational State Transfer*" und wird von Fielding [11] als einen Stil der Architektur für verteilte hypermediale Systeme (wie das World Wide Web, das man heutzutage kennt) genutzt. Kombiniert man die beiden Definitionen von Rest und API, erhält man die Definition: Eine API, die den Stil einer REST-Architektur folgt.

Zweck

Der Hintergedanke für REST ist die Optimierung eines verteilten Systems, das REST-mäßig verwaltetet ist, in folgenden Bereichen [12]:
(1) *Performance*, (2) *Skalierbarkeit der Interaktion von Komponenten*, (3) *Einfachheit des Interfaces*, (4) *Modifizierbarkeit der Komponenten*, (5) *Portabilität*, (6) *Betrieblichkeit/-sicherheit*, (7) *Sichtbarkeit*.

2.2.2 Constraints

Fielding erläutert 2 Arten einer Systemdefinition. Ersteres wird beschrieben als ein blankes unbeschriebenes Blatt/Whiteboard, das zu Beginn nicht weißt, dass ein System gebaut wird sowie die Nutzung bekannter Komponenten, bis der Bedarf davon besteht. Die Zweite Variante ist ein komplettes Set, das den gesamten Bedarf eines Systems abdeckt und Constraints schrittweise den Komponenten hinzugefügt wird, bis alles reibungslos miteinander funktioniert.
Von den genannten Angehenweisen, befolgt REST das Letztere. Beim Bau einer Rest-Architektur, definiert man am Anfang einen **null-state** - ein System, das keine Constraints beinhaltet und zwischen den Komponenten noch keine Unterschiede gibt. Die Constraints werden dann nach und nach hinzugefügt [12].

Client-Server

Der erste Constraint, ist der am häufigsten genutzte Constraint in einer netzwerkbasierten Architektur, worin Clients und Server eine separate Rolle haben. Dieser Constraint erlaubt dem Client und Server die Flexibilität sich unabhängig voneinander zu entwickeln, sodass es kein Effekt auf den Code beider Seiten hat [10, 12].

Stateless

Mit dem Stateless-Constraint ist es für den Web-Server nicht erfordlich den Status der Client-Applikation zu merken. Zusätzlich muss der Client für die Interaktion mit dem Web-Server alle für ihn relevanten Informationen bei einer Anfrage mitliefern. Indem der Client die Komplexität der Mitteilung des Applikation-Statuses bewältigt, kann der Web-Server eine große Menge von Clients dienen[10].
Folgende Bereiche werden durch das Constraint verbessert [12]:

1. Visibility: Da alle nötigen Informationen innerhalb der Anfragen sind, erleichtert es die Überwachung des Systems

2. Scalability: Der Server kann Ressourcen schneller freigeben, weil keine Daten zwischen den Anfragen gespeichert werden müssen.

3. Reliability: Ein *stateless* System muss bei einem Fehlschlag nur die Applikation wiederherstellen

4. Einfachere Implementierung: Das System auf der Seite des Servers wird simpler, weil man nicht mit gespeicherten Zustandsdaten über mehrere Server arbeiten muss.

Auch wenn das Constraint viele Vorteile bietet, können geringe Overheads in Anfragen mit wiederholten Zustandsdaten den Netzwerk-Verkehr beeinflussen. Abhängig von der Art der System-Implementierung und der Anzahl wiederholter Statusinformationen, kann es ein kritisches Problem darstellen [12].

Cacheable

Das Cacheable-Constraint ist eines der wichtigsten Constraints für einen Web-Server [10]. Durch das Constraint werden die Response-Daten zwischengespeichert und macht sich bemerkbar auf der Seite des Clients [12]. Weitere Vorteile wie die Steigerung der Verfügbarkeit und Zuverlässigkeit der Applikation sind das Resultat davon[10]. Der einzige Nachteil ist, das man durch die mangelnden Caching-Regeln veraltete zwischengespeicherte Daten erhält [12].

Uniform Interface

Dieser Constraint ist eines der Hauptcharakteristiken von REST und zugleich eines der größten Vorteile, wenn man es mit anderen Alternativen vergleicht [12]. Durch die Vereinheitlichung der Komponenten - Clients, Server und netzwerkbasierte Vermittler - erleichtert man den Clients die Interaktion mit dem System[12, 10]. Ein weiterer großer Vorteil ist, dass man mit standardisierte und einheitliche Services, die Implementierungen der Clients effektiv vereinfacht, indem man klare Regeln definiert, die sie folgen müssen[12].

Ein Nachteil des Constraints sind Performance-Beeinträchtigungen, falls es eine optimiertere Form der Kommunikation existieren soll.

Layered System

Da REST für das Internet entworfen wurde und dort eine riesige Menge an Verkehr herrscht, wird das Constraint *Layered System* eingeführt, dass dies bewältigen soll. Es werden für diesen Zweck die Komponenten in Schichten augeteilt. Dabei dürfen die Schichten nur die Funktionalitäten der unterliegenden Schicht nutzen und das Ergebnis nur mit der Schicht über sich mitteilen. So wird die Komplexität des Systems vereinfacht und die Komponenten-Kopplungen werden so im Stande gehalten.
Ein Nachteil des Constraints ist, dass wenn man es bei einem kleinen System anwendet, verlangsamt man es durch die Interaktion der verschiedenen Schichten[12].

Code-on-Demand

Der Constraint erlaubt dem Client den Code, das vom Server bereitgestellt wird, zu downloaden. Jedoch muss der Client den Code verstehen können, um es auszuführen. Aus diesem Grund ist es das einzige Constraint, das optional ist [12, 10].

2.2.3 RESTful HTTP-Services

Die REST-Architektur ist Ressourcen-orientiert [13], woraus man schließen kann, dass Ressourcen ein Hauptbestandteil von REST ist. Einige Beispiele für Ressourcen sind:

1. Ein Image - in dem HTML-Tag *<img src="image.png">* wird mit dem Attribut *src* auf das Image verwiesen

2. ein Element aus der Datenbank, dass durch die ID referenziert wird

3. Eine Webseite, das man mit der URL referenziert

Für diese Ressourcen gibt es sogenannte RESTful Http-Services (auch bekannt als *Http-Verbs*), die eine bestimmte Aktion ausführen. Diese Aktionen werden auch *CRUD*-Aktionen genannt (create, read, update und delete), die abhängig von den Ressourcen, implementiert und bereitgestellt werden. Diese CRUD-Funktionen sollten von jedem ressourcen-orientierten System bereitgestellt werden. [12]
Tabelle 2.1 gibt eine Übersicht über 6 häufig genutzte Http-Verbs an.

Http-Verb	Aktion	CRUD-Aktion
GET	Zugriff auf Ressourcen in read-only-Modus	READ
POST	Sendet neue Ressourcen an den Server	CREATE
PUT	Aktualisieren von Ressourcen	UPDATE
DELETE	Löscht Ressourcen	DELETE
HEAD	Schickt Anfrage auf Existenz einer Ressource, aber keine Darstellung der Ressource wird zurückgegeben	keine CRUD-Aktion
OPTIONS	Gibt die Liste von Http-Verbs einer Ressource zurück.	kein CRUD-Aktion

Tabelle 2.1: HTTP-Verbs und deren Funktion [12]

3 Qualitätsbewertung

Viele der Literaturen in der Qualitätsbewertung einigen sich auf den Slogan "fitness for purpose", d.h. die Qualität eines Objektes wird durch den Grad der Unterstützungsfähigkeit eines gewissen Zweckes bestimmt. Péter Király [14] deutet auf 3 weitere Definitionen für den Slogan "fitness for purpose" hin:

- Erfüllung einer Spezifikation oder eines angegebenen Resultats
- Messung an das, was man als Ziel einer Einheit ansieht
- Erreichen der Ziele eines Instituts

Aus den Defintionen konnten dann 2 Schlussfolgerungen gemacht werden [14]:
(1)*Die Qualität eines Objektes ist abhängig vom Kontext und ist somit kein absoluter Wert, d.h. die Qualität ist ein Indikator eines Objekts Güte, die angibt wie hilfreich es ist, das/die Ziel/e in einem bestimmten Kontext zu erreichen.*
(2)*Die Qualität ist ein facettenreicher Wert. Es gibt viele Aspekte, die man beachten muss.*

Die zweite Schlussfolgerung weist auf einen sehr wichtigen Fakt hin! Die Qualität betrachtet verschiedene Aspekte und nicht nur ein einzelnes. Der Begriff "*Aspekte*" wird in dieser Arbeit auch als "*Dimension*" bezeichnet.
Das Ziel in diesem Kapitel ist es, herauszufinden, welche Dimensionen bereits existieren und zu dem ORKG-Dienst[1] (werden im 2. Abschnitt definiert) passen, und die Definition der Messtechniken (*Metriken*). Im ersten Abschnitt des Kapitels wird auf die Priorisierung der Dimensionen eingegangen, das im darauffolgenden Abschnitt bei der Definition der Dimensionen und Metriken helfen soll.

[1] `https://www.orkg.org/orkg/`, Zugriff 04.06.2022

3.1 Priorisierung

Wie bereits erwähnt, gibt es verschiedene Dimensionen, die betrachtet werden müssen, um die Qualität eines Datensatzes zu bestimmen. Es gibt jedoch sehr viele Dimensionen und es können nicht alle gedeckt werden. Aus diesem Grund wird in diesem Abschnitt die Priorisierung aus der "*FAIR Data Maturity Model*"[15] vorgestellt und darin unterteilte man die Indikatoren der Dimensionen (Aspekte) in 3 Prioritätsstufen:

1. *Essentiell* - Indikator mit der höchsten Wichtigkeit. Ohne den Indikator ist der weitere Fortschritt der Bewertung nicht gewährleistet.

2. *Wichtig* - Es ist vom Grad der Wichtigkeit her nicht so hoch wie essentiell, aber das Einbeziehen dieses Indikators kann einen erheblichen Beitrag zum Ergebnis liefern.

3. *Nützlich* - Verzichtbare Indikator: Es ist gut, wenn man es bei der Bewertung einschließt, ist jedoch nicht nötig.

Klassifiziert man die Prioritätsstufen aller Indikatoren, erkennt man, dass Dimensionen mit einer hohen Indikatorenzahl eine hohe Gesamt-Priorität haben. Das führt zu einem Konflikt, in der Dimensionen mit einer niedrigen Indikatoren-Zahl dieselbe Gesamt-Priorität haben kann wie Dimensionen mit einer hohen Indikatoren-Zahl.

Angenommen die Prioritätsstufen ausgedrückt in Zahlen ergeben *essentiell := 3*, *wichtig := 2* und *nützlich := 1*. Kleine Dimensionen dessen Indikatoren alle die Prioritätsstufe "*essentiell*" haben, besitzen eine Gesamtpriorität dessen Wert einer 3x so großen Dimension entspricht, in der die Prioritätsstufe aller Indikatoren in "*nützlich*" gestuft ist. Jedoch kann man auf die Indikatoren der 3x so großen Dimension verzichten. Aus diesem Grund ist es angemessener die Dimensionen nach der Anzahl essentieller Indikatoren absteigend zu sortieren, weil ohne essentielle Indikatoren das weitere Verfahren der Bewertung sonst nicht gewährleistet werden kann. Dasselbe Verfahren wird mit der nächstniedrigeren Prioritätsstufe (in dem Fall "*wichtig*") bei Dimensionen angewendet, dessen

Anzahl essentieller Indikatoren gleich sind. Dieses Sortierverfahren kann so lange fortgesetzt werden bis es keine niedrigere Prioritätsstufe gibt.

Die Klassifizierung der Indikatoren in Prioritätsstufen ist jedoch community-abhängig. Da die Voraussetzungen und somit auch die Indikatoren sich von Community zu Community unterscheiden und wofür es in der Praxis angewendet wird. Darum wird empfohlen sich im Voraus Gedanken über die Voraussetzungen zu machen und wie es in der Anwendung helfen soll [15].

3.2 Dimensionen & Metriken

In dem Artikel *Linked Open Data* [16] (LOD) wurde ein großer Aufwand in einer Befragung und Filterung vieler Artikel basierend auf deren Titel investiert. Dies lieferte 118 relevante Artikel und als Ergebnis einer weiteren Analyse, wurden aus der Menge insgesamt 21 Artikel rausgesucht, in der man diese Ähnlichkeiten fand:

- häufige Verwendung von Terminologien im Bezug auf Qualität der Daten
- 23 verschiendene Dimensionen und deren formalisierten Definition
- Metriken der Dimensionen mit Unterscheidung zwischen subjektiv und objektiv
- Vergleich der Tools um die Qualität der Daten zu messen

Die Dimensionen klassifizierte man in dieser Arbeit in 6 Kategorien: *Accessibility, Intrinsic, Trust, Dataset dynamicity, Contextual und Respresentational*. In der Tabelle 3.1 wird gezeigt wie Zaveri [16] die Dimensionen kategorisiert und welche Prioritätsstufe ich den Dimensionen zugewiesen habe. Im Abschnitt, in der die Metriken definiert werden, wird in jeder essentiellen Dimension, eine Bemerkung gegeben, die erläutert, warum ich es als essentiell gesehen habe.

3.2.1 Dimensionen

In der Tabelle 3.1 werden den Dimensionen eine Kategorie und Prioritätsstufe zugewiesen. Es werden hierfür nur die essentiellen Dimensionen erläutert. Da sie die wichtigsten Qualitäts-Aspekte darstellen und als Basis der Qualitätsbewertung dienen sollen. Unter dem Bemerkungsabschnitt werden einige Dimensionen aus der Tabelle 3.1 genannt, die in Betracht gezogen wurden, aber nicht als essentiell eingestuft sind.
Zaveri [16] konnte eine häufige Nutzung der Dimensionen *Believeability, Consistency, Currency, Completeness, Accuracy und Availability* erkennen. Jedoch ist die Wahl dieser Dimension nicht die Norm für die Qualitätsbestimmung. Dennoch werden aus den genannten Dimensionen *Availability*, *Completeness* und *Accuracy* als essentiell eingestuft. Die Dimensionen Objectivity und Understandability habe ich zu der Liste der essentiellen Dimensionen angehangen.

Availability

Definition: *Verfügbarkeit der Indikatoren für eine Metrik [17]. Angepasst auf die Use-Cases: Existenz eines Idikators, sodass mindestens eine Metrik einer Dimension berechnet werden kann.*

Accuracy

Hildebrand [18] nennt diese Dimension auch "*free of error*" und erläutert ein Teil-Aspekt der Dimension. Furber [19] ergänzt die Dimension mit einer Unterteilung in semantische und syntaktische Accuracy.
Definition: *Werte der Indikatoren repräsentieren den Stand der realen Welt (semantisch) und befolgen ein bestimmtes Schema (syntaktisch), d.h. frei von Syntaxfehler [16].*

Objectivity

Zaveri [16] übernahm die Definition von Pipino [20]: *Grad der Neutralität von Informationen*

Completeness

In Mendes Arbeit [21] unterscheidet man Completeness in *Schema-Level* und *Data-Level*. Im *Schema-Level* bezieht man sich im Datensatz auf die Vollständigkeit der nötigen Attribute einer Aufgabe. Im *Data-Level* dagegen bezieht es sich auf die Vollständigkeit aller nötigen Objekte für eine Aufgabe. **Definition**: *Grad der Präsenz aller nötigen Informationen in einem Datensatz. [16]*

Understandability

Zaveri [16] kombiniert die Definitionen [20, 22, 17], sodass man es versteht als: *Grad der Verständlichkeit, d.h. für den Informationsverbraucher ist die Information leicht zu begreifen und es besitzt keine Mehrdeutigkeiten.*

Kategorie	Dimension	Priorität
Accessibility	Availability	essentiell
	Licensing	(neu, LOD) wichtig
	Interlinking	(LOD) nützlich
	Security	(LOD) nützlich
	Performance	nützlich
Intrinsic	Accuracy	essentiell
	Consistency	wichtig
	Conciseness	(LOD) wichtig
Trust	(LOD) Reputation	nützlich
	Believability	wichtig
	Verifiability	wichtig

Kategorie	Dimension	Priorität
	Objectivity	essentiell
Dynamicity	Currency	nützlich
	Volatility	(LOD) wichtig
	Timeliness	(LOD) nützlich
Contextual	Completeness	essentiell
	Amount-of-Data	nützlich
	Relevancy	wichtig
Representation	Representational-Conciseness	nützlich
	Respresentational-Consistency	wichtig
	Understandability	essentiell
	Interpretability	wichtig
	Versatility	wichtig

Tabelle 3.1: Kategorisierung Dimsenionen

Bemerkung

Alle Dimensionen, dessen Prioritätsstufe "*nützlich*" entsprechen, sind verzichtbar und werden hier nicht weiter behandelt. Für das weitere Interesse an Dimensionen, die hier nicht abgedeckt werden, wird Zaveris Arbeit [16] empfohlen, in der man eine sehr ausführliche Beschreibung aller Dimensionen aus der oberen Tabelle 3.1 erhält.

Licensing:*Zulassung für die Nutzung des Datensatzes unter bestimmten Bedingungen [16].*
In dieser Arbeit werden Provider gewählt, in der die Nutzung der Datensätze nicht eingeschränkt sind.

Conciseness: *Entität besitzt nur einzigartige Attribute, d.h. es existieren keine*

weiteren Attribute, die die selbe Information repräsentieren [16].
Da nur gewisse Meta-Felder im ORKG-Dienst betrachtet werden und sie alle einzigartige Informationen bieten, sind sie für die Qualitätsbewertung bereits einzigartige Attribute.

Consistency: *Für die Dimension soll es keinen Widerspruch zwischen 2 Werten existieren, d.h. in einem Datensatz darf kein Indikator (Wert eines Metafeldes) ein anderes aufheben [16].*
Wie bereits in Conciseness beschrieben, sind die Meta-Felder einzigartig. Die ausgewählten Metadaten, würden mit anderen Metadaten innerhalb der ausgewählten Metadaten-Menge zu keinem Widerspruch führen.

Believeability:*Grad in der Informationen als korrekt, wahr, echt und glaubwürdig angesehen wird [16].*
Man könnte diese Dimension in Kombination mit *Completeness* verwenden, sodass man auf die Existenz der nötigen Indikatoren (Titel, Inhalt und URL) überprüft (wird in dieser Arbeit jedoch nicht so umgesetzt). Ich kombiniere es Objectivity.

Verifiability:*Grad der Korrektheit des Datensatzes die ein Datenverbraucher bewertet [16].*
Zaveri [16] hat 3 Metriken genannt. Aber die Indikatoren sind abhängig vom Provider und sie sind nicht innerhalb der ausgewählten Menge von Metadaten für den ORKG-Dienst.

Volatility:*Anzahl der vorgenommenen Änderungen [16].*
Diese Dimension kann dabei helfen herauszufinden, ob man den Informationen aus dem Datensatz glauben schenken kann. Die Anzahl der Änderungen deutet auf Fehler im Datensatz hin. Andere Dimensionen (wie Objectivity) bieten auch Metriken, die die Glaubwürdigkeit des Datensatzes bewerten kann.

Representational-Consistency:*Format und Struktur eines Datensatzes entspricht dem des vorherigen Ergebnisses und den von anderen Providern [16].*
Diese Dimension wird nicht gemessen und soll nur bei der Wahl der Provider helfen. Da es ein einheitliches Format bereitstellt und der Zugriff auf die

Informationen erleichtert wird.

Interpretability: *Bestimmte Notation der Daten und Darstellung der Information entspricht der technischen Anwendung des Nutzers [16].*
Das kann hilfreich für das weitere Verfahren des Datensatzes sein. Jedoch ist es nicht essentiell, da man die Information in einem Zwischenschritt bearbeiten kann, sodass es für den Informationskonsumer geeignet ist.

Versatility: *Verfügbarkeit der Daten in internationaler Form, Verfügbarkeit einer alternativen Darstellung und die Verfügbarkeit verschiedener Zugriffsmethoden [16].*
Das ist eine Dimension, die bei der Auswahl von Providern helfen kann.

3.2.2 Metriken

Es werden in diesem Abschnitt für jede essentielle Dimension eine Metrik erläutert, die im 6.Kapitel implementiert werden. Da die These sich auf den ORKG-Dienst bezieht, werden hauptsächlich deskriptive Metadaten für die Metriken genutzt, was man in den Abbildungen A.1 A.2 erkennen kann.

Availability

Die Dimension *Availability* ist die Basis jeder Metrik. Denn die Existenz eines Indikators einer Metrik muss gegeben sein, um die Metrik auch durchführen zu können. Aus diesem Grund gibt es in dieser Arbeit keine explizite Bewertung dieser Dimension, d.h. es werden in den Metriken aller anderen Dimensionen auf die Availability der Indikatoren geprüft und beeinflusst so indirekt das Ergebnis der Qualitäts-Bewertung.

Completeness

Für diese Dimension werden bestimmte Meta-Felder, dessen Existenz für die Arbeit relevant sind (*siehe Anhang Abbildung A.1, A.2*), betrachtet und bildet die totale Anzahl relevanter Meta-Felder m. Die Existenz eines relevanten Meta-Feldes x_i entspricht dem Wert **1**. Die Metrik dieser Dimension untersucht den Datensatz auf die Existenz der relevanten Meta-Felder und bildet eine Summe daraus. Der Completeness-Grad wird durch die Beziehung zwischen existierende relevante Meta-Felder und Menge aller relevanten Meta-Felder *Relevant* gebildet, was diese Formel wiedergibt:

$$m = |Relevant| \in N \qquad Relevant = \{x | x \in Datensatz \cap x \text{ ist relevant}\}$$

$$\frac{\sum_{i=1}^{n} f(x_i)}{m} \qquad f(x_i) = \begin{cases} 1, & \text{falls } x_i \in Relevant \cap x_i \in Datensatz \\ 0, & \text{sonst} \end{cases}$$

Bemerkung: Für den ORKG-Dienst werden gewisse Meta-Felder (deskriptive Metadaten) betrachtet. Solche Felder sind wichtig, um den Eintrag der Publikation zu finden [9]. Eine große Anzahl ausgefüllter relevanter Meta-Felder erleicht somit die Suche und das Auffinden der Publikation.

Accuracy

Die Metrik der Dimension *Accuracy* befasst sich mit Indikatoren, die die Fakten aus der realen Welt repräsentieren und frei von Syntaxfehler sind [16].
Auswahl der Meta-Felder für diese Dimension mit folgenden Eigenschaften aus der realen Welt:

- DOI einzigartig, folgt einem Format, das immer mit "10." beginnt
- Publikationsjahr ganzzahlig, ist niemals größer als das aktuelle Jahr
- Publikationsmonat ganzzahlig, zwischen 1-12

Bemerkung: Die Metrik bezieht sich auf den ORKG-Dienst. Im Gegensatz zu den restlichen Feldern (Abbildung A.1, A.2), repräsentieren die 3 Felder Fakten aus der realen Welt. Mit der Accuracy Dimension vermeidet man Syntax-Fehler im ORKG-Dienst und kann somit einen Aufwand in der Verwaltung minimieren.

Objectivity

Eines der Metriken für diese Dimension, die Bizer [17] beschrieb, ist die Untersuchung eines Faktes in verschiedenen Quellen, sodass man eine Bestätigung des Faktes erhalten kann.
Es werden für diese Metrik alle Meta-Felder ausgewählt, die in jedem Datensatz den exakt gleichen Wert haben. Man sollte bei der Auswahl der Meta-Felder darauf achten, dass sie immer in allen Datensätzen vorkommen.
Aus allen relevanten Felder werden diese (deskriptiven Metafelder) rausgesucht:

- DOI Identifikationsmöglichkeit: eindeutig
- Titel Indentifikationsmöglichkeit: uneindeutig - da man eine Arbeit durch den Titel erkennen kann und mehrere Arbeiten denselben Titel haben können, ist der Titel nicht einzigartig und somit nicht eindeutig.
- Publikationsjahr bei einer Publikation sollte mindestens das Publikationsjahr angegeben werden
- Authoren Publikation hat mindestens 1 Author

Bemerkung: Mit der Metrik kombiniert man die Dimension Objectivity und Believability. Je mehr Quellen man zur Verfügung hat, desto mehr Metadaten-Vergleiche kann man durchführen. Eine hohe Anzahl von Bestätigungen verschiedener Quellen deutet auf ein höheres Vertrauen der bereitgestellten Informationen.

Understandability

In dieser Dimension bezieht sich die Metrik nur auf die Authorennamen. Da Authorennamen zu den deskriptiven Metadaten gehören, erleichtert es das Auffinden, Identifizieren und Suchen von Publikationen. Es kommt häufig vor, dass man Aliase in Authorennamen auffindet, womit eine eindeutige Identifizierung erschwert wird, weil Authoren denselben Nachnamen und dieselben Aliase haben können.

4 User-Stories & Use-Case Tabellen

Dieses Kapitel soll als Grundlage für die Implementierung dienen. Der erste Abschnitt definiert User-Stories, die in Pflicht und Optional gegliedert werden. Die *Pflicht-User-Stories* zählen Aspekte auf, die umgesetzt werden müssen, um eine Basis zu gründen, sodass das Programm angewendet werden kann. In den *Optionalen* dagegen, kann man dann entscheiden, ob man die dort definierten Funktionalitäten umsetzt oder nicht.
Der letzte Abschnitt dieses Kapitels beschäftigt sich mit der Erstellung von Use-Case-Tabellen. Es werden dabei 2 Hauptfälle betrachtet, indem der Nutzer mit dem Programm interagiert.

4.1 User-Stories

4.1.1 Pflicht

U1 **Als** *Forscher*
möchte ich *eine Schnittstelle haben, worin der Abruf der Ressourcen und Vergleich der Datensätze kombiniert sind*,
sodass ich *nicht mehrere Schnittstellen nutzen muss.*

U2 **Als** *Forscher*
möchte ich *einen Datensatz über die DOI abrufen*,
sodass ich *nicht überprüfen muss, ob der Datensatz von der gesuchten Publikation ist.*

U3 **Als** *Forscher*
möchte ich, *dass die Metadatensätze von mehreren Quellen stammen und verglichen werden*,
sodass ich *mir die manuelle Arbeit der Suche und Vergleich sparen kann.*

U4 **Als** *Forscher*
möchte ich *den qualitativ hochwertigsten Metadatensatz erhalten*,
sodass ich *den besten Datensatz für die weitere Untersuchung nutzen kann.*

U5 **Als** *Forscher*
möchte ich *eine Dokumentation zu dieser Schnittstelle haben*,
sodass ich *nicht in den Code schauen muss, um eine nähere Erläuterung der einzelnen Funktionen zu erhalten.*

4.1.2 Optional

OU1 **Als** *Forscher*
möchte ich *einen Datensatz über den Publikationstitel abrufen*,
sodass ich *nach einem Datensatz einer Arbeit suchen kann, dessen Titel interessant ist, und ich nicht die Zeit habe nach der DOI zu suchen.*

OU2 **Als** *Forscher*
möchte ich *nicht länger als 3 Sekunden auf den Datensatz warten*,
sodass ich *nach mehr Datensätze innerhalb eines gewissen Zeitraums abfragen kann.*

OU3 **Als** *Forscher*
möchte ich *eine qualitativ absteigend sortierte Liste der Datensätze mit deren Bewertungsstatistik haben*,
sodass ich *sehen kann, welche Quellen Datensätze zurückgeliefert haben und welche Bewertungen deren Datensätze in den einzelnen Dimensionen erhielten.*

4.2 Use-Cases

Use-Case	REST-API Anfrage mit DOI
Hauptakteur	Forscher/in
Systemgrenzen	REST-API Anfragen im local host
Interessen	qualitativ hochwertige Metadaten erhalten
Voraussetzungen	1. im Besitz eines internet-verbindungsfähiges Gerät 2. Zugang zum Internet
Garantie	Es wird entweder ein Metadatensatz einer Publikation in JSON-Format oder nichts ausgegeben
Erfolgsfall	hochwertigster Metadatensatz wird zurückgeliefert
Auslöser	Eingabe einer gültigen DOI für Query-Parameter *doi* in Webbrowser und *Enter* drücken
Beschreibung	1. Eingabe einer gültigen DOI für Query-Parameter *doi* in Webbrowser und *Enter* drücken 2. Programm sendet Anfrage an Endpunkt der Provider, um Ressource abzufragen 3. Provider sendet Response mit Ressourceninhalt an Programm 4. Programm filtert Responses nach Status-Code 200 5. Programm bewertet Metadatensatz aus den Responses nach den definierten Metriken 6. Programm liefert Datensatz mit höchster Qualität zurück
Erweiterung	1.1 Eingabe von mehreren DOIs 2.1 Programm ruft mehrere Ressourcen (durch mehrere DOIS) ab

Use-Case	REST-API Anfrage mit Publikationstitel
Hauptakteur	Forscher/in
Systemgrenzen	REST-API Anfragen im local host
Interessen	qualitativ hochwertige Metadaten erhalten
Voraussetzungen	1. im Besitz eines internet-verbindungsfähiges Gerät 2. Zugang zum Internet
Garantie	Es wird entweder ein Metadatensatz einer Publikation in JSON-Format oder nichts ausgegeben
Erfolgsfall	hochwertigster Metadatensatz wird zurückgeliefert
Auslöser	Eingabe eines Publikationstitels für Query-Parameter *query* in Webbrowser und *Enter* drücken
Beschreibung	1. Eingabe eines Publikationstitels für Query-Parameter *query* in Webbrowser und *Enter* drücken 2. Programm sendet Anfrage an Endpunkt der Provider, um Ressource abzufragen 3. Provider sendet Response mit Ressourceninhalt an Programm 4. Programm filtert Responses nach Status-Code 200 5. Programm filtert Datensatz mit höchster Ähnlichkeit zum eingegebenen Titel raus 6. Programm bewertet Metadatensatz aus den Responses nach den definierten Metriken 7. Programm liefert Datensatz mit höchster Qualität zurück
Erweiterung	5.2 Programm überspringt den Schritt und geht direkt in die Bewertung der Datensätze 6.2.1 Programm bewertet alle Datensätze 6.2.2 hochwertigster Datensatz einer DOI wird an einer Liste von hochwertigen Datensätzen angehangen 7.2.1 Programm zeigt eine Liste von hochwertigen Datensätzen verschiedener DOIs an 7.2.2 Nutzer wählt einen Datensatz aus 7.2.3 Programm liefert den gewählten Datensatz zurück

5 Provider-Analyse

Für die Aufgabe der Qualitätsbewertung von Metadaten wissenschaftlicher Publikationen, werden nach Anbieter (Provider) für Literaturrecherche gesucht. Die Suche nach den Anbietern fand über Google statt und begann mit den bekanntesten und größten Anbieter für Literaturrecherche, Google Scholar. Es wurde dann viel mit dem String "scholar" und "provider" für die Suche ausprobiert, um neue Ergebnisse zu finden. Dabei ging man auf Vorschläge einer ähnlichen Suche ein, um eine größere Reichweite von Ergebnissen zu erhalten. Es wurde eine große Menge von Anbieter für Literaturrecherche gefunden. Aus dieser Menge wurden 10 bekannte Anbieter rausgesucht und in der Tabelle 5.1 miteinander verglichen.

5.1 Auswahl-Kriterien

API

Auf den beiden Use-Cases bezogen, benötigt der Dienstanbieter eine Schnittstelle, in der man die Metadaten zu den Publikationen abrufen kann. Andernfalls erschwert es den Prozess Metadaten zu erhalten. Eine ideale Lösung ist eine REST-API.

DOI

Die Schnittstelle, die realisiert wird, arbeitet mit einem gültigen *Digital Object Identifier* (DOI). Die Schnittstelle der Dienstanbieter sollte eine Möglichkeit bieten, mit der man über die DOI die Metadaten der gewünschten Publikation abrufen kann.

JSON-Response

Das dritte Kriterium ist eine persönliche Präferenz. Die Datensätze mit den Metadaten einer Publikation sollte in einem JSON-Format dargestellt werden. Manche Provider wie Arxiv und PubMed liefern Datensätze im XML-Format, dessen Lesbarkeit meiner Meinung nach schlechter ist als die mit einem JSON-Format.

Anzahl Publikationen

Ein wichtiger Aspekt ist die Anzahl der Publikationen, die ein Anbieter besitzt. Der Gedanke hierhinter ist, dass man so viele Anfragen wie nur möglich abdecken möchte, sodass Ergebnisse geliefert werden können. Dieser Aspekt spielt eine große Rolle in der Qualitätsdimension *Availability*. Denn ein Service wird nur dann genutzt, wenn es auch Ergebnisse liefert. Je öfter der Fall, in der kein Ergebnis zurückgeliefert wird, eintretet desto höher steigt die Wahrscheinlichkeit, dass der Service nicht in Anwendung kommt.

Kostenlos

Kostenlose Anbieter sind meistens Open Access, sodass man keine Gebühren für die Ressourcen-Zugriffe zahlen muss und die Informationen aus den Datensätzen genutzt werden können. Somit kann man den Nutzern einen kostenlosen Service

bieten.

5.2 Auswahl

Nachdem die Auswahl-Kriterien definiert wurden, werden in der Tabelle 5.1 10 bekannte Anbieter in absteigender Reihenfolge basierend auf deren Publikationen-Zahl gelistet. Alle Anbieter wurden auf die Auswahlkriterien (jeweils auf dessen offiziellen Web-Seite) untersucht.

Die Auswahl der Anbieter trifft auf diejenigen, die in allen Auswahl-Kriterien einen Wert besitzen. Diese Bedingung erfüllen die Provider *(1) Semantic Scholar*, *(2) CrossRef*, *(3) Dimensions* und *(4) DataCite*. Jedoch ist der Service von *Dimensions* kostenpflichtig, was in dieser Arbeit ein Nachteil gegenüber kostenlose ist, und gehört somit nicht in die engere Auswahl. Übrig bleiben nur noch *(1) Semantic Scholar*, *(2) CrossRef* und *(3) DataCite*. Die Metadaten aus den Datensätzen dieser 3 Anbieter werden für die Qualitätsbewertung dienen.

Provider Liste					
Provider-Name	API	DOI	JSON-Responses	Anzahl Publikationen	Kostenlos
Google Scholar	Nein	-	-	389 Mio.	Ja
Semantic Scholar	Ja	Ja	Ja	über 200 Mio.	Ja
Web of Science	Ja	-	-	171 Mio.	Nein
CrossRef	Ja	Ja	Ja	135 Mio.	Ja
ResearchGate	-	-	-	135 Mio.	Ja
Dimensions	Ja	Ja	Ja	106 Mio.	Nein
Scopus	Ja	-	Ja	über 83 Mio.	Ja
PubMed	Ja	-	Nein	über 34 Mio.	Ja
DataCite	Ja	Ja	Ja	über 16 Mio.	Ja
Arxiv	Ja	-	Nein	2 Mio.	Ja

Tabelle 5.1: Tabelle gefüllt mit Provider und was sie bieten

Provider Bemerkungen

Microsoft Academic gehörte ursprünglich zur Provider-Liste. Jedoch hat Microsoft am 04.05.2021 in einem Blogpost[1] veröffentlicht, dass sie den Service von Microsoft Academic bis Ende 2021 weiterhin betreiben werden und es danach abstellen, sodass die Nutzer auf andere Dienste umsteigen können.

Aus der Liste ist Google Scholar mit 389 Millionen Publikationen mit Abstand der größte Dienstanbieter für Literaturrecherche. Ohne viel nachzudenken wäre Google Scholar auf der Liste von Provider für die Qualitätsbestimmung von Metadaten, **aber** Google Scholar bietet selber keine Schnittstelle an, welches Metadatensätze von Publikationen zurückliefert.

Der Dienstleister *DataCite* bietet neben der REST-API noch einen alternativen Zugriff über GraphQL, als Query Sprache für deren API an. Die Besonderheit bei GraphQL ist, dass man den HTTP-Verb *POST* nutzt und man im Body einer POST-Anfrage definieren kann, welche Metadaten interessant sind und zurückgeliefert werden sollen.

[1] `https://www.microsoft.com/en-us/research/project/academic/articles/microsoft-academic-to-expand-horizons-with-community-driven-approach/`, zugegriffen am 10.05.2022

6 Implementierung

Das Ziel in diesem Kapitel ist die Entwicklung einer REST-API sowie die Realisierung aller Metriken der essentiellen Dimensionen und die Umsetzung aller User-Stories (definiert in Kapitel 4). Es wird für die Implementierung eine moderne, sehr beliebte Programmiersprache (:= *Python*) genutzt, die eine reiche Auswahl an Libraries und Frameworks besitzt und eine große, hilfreiche und aktive Community hinter sich hat.
Da es sich in diesem Kapitel um die Implementierung handelt, sind Programmierkenntnisse hilfreich, wobei grundlegende Kenntnisse der Programmiersprache *Python* voraussgesetzt werden. Für die Implementierung wird die Version *Python 3.10.1* genutzt, um die Kompatibilität aller genutzten Libraries und Frameworks zu gewährleisten.

6.1 U1: Schnittstelle REST-API

Das Erste, was realisiert werden muss, ist die Schnittstelle. Darüber erfolgen alle Abfragen und Qualitätsbewertungen.
Mithilfe des Web-Frameworks *FastAPI* wird eine REST-API gebaut, womit der Nutzer über einen beliebigen Web-Browser die Metadaten einer Publikation abrufen kann. Dabei soll der Abruf über die Suchzeile erfolgen, dass in einem Pfad stattfindet, das dem Nutzer bevorzugten Suchart (DOI := "*/search-doi/*" oder Publikationstitel := "*/search-title/*") entspricht.

FastAPI Haupt-Merkmale

Aus der offiziellen Seite[1] von FastAPI werden folgende Merkmale aufgelistet, die für die Nutzung des Web-Frameworks sprechen:

1. *schnell* - mit dem Namen wird es bereits deutlich gemacht, dass es schnell ist. Laut deren Seite ist FastAPI (durch die Nutzung der Frameworks *Starlette* und *Pydantic*) auf Augenhöhe mit *NodeJS* und *GO*.

2. *schnelle Codierung* - erlaubt eine Steigerung von 200% bis 300% der Feature-Entwicklung

3. *weniger Bugs* - 40% der von Menschen (-Entwickler) verursachten Fehler werden reduziert

4. *einfach* - einfache Nutzung und Lernfähigkeit führt zu geringerem Gebrauch der Dokumentation

5. *kurz* - reduziert Code Duplikationen

6. *robust* - Code in Produktions-Version mit Hilfe der automatisch interaktiven Dokumentation

7. *standard-basierend* - basiert auf die "*open standards for API*": OpenAPI und JSON Schema

6.1.1 U2: API-Anfragen mit DOI

Die REST-API soll die Anwendung von Zugriffsoperationen (in dem Kontext GET) auf eine bestimmte Ressource ermöglichen. Es werden nach den Use-Cases nur Lese-Zugriffe benötigt, weil die Informationen von den Ressourcen gelesen wird und man es nicht modifizieren möchte. Der Code-Snippet 6.1 beschreibt die Definition einer solchen Lese-Operation:

```
1 @app.get('/search-doi/')
```

[1] `https://fastapi.tiangolo.com` Zugriff 20.05.2022

```
2  async def doiRequests(doi: str):
3      req = AsyncRequest([doi])
4      await req.doiGetMeta()
5      if len(req.results) == 0:
6          raise HTTPException(status_code=404, detail="Item not found
       ")
7      # bestimme die Qualitaet der Datensaetze
8      assessment = DatasetAssessment(req.results)
9      assessment.computeQuality(doi)
10     return assessment.quality_result[0][1]
```

Listing 6.1: REST-API GET-Request mit DOI

Es wird in der ersten Zeile FastAPI mitgeteilt, dass die Funktion, die jetzt folgt, für eine GET-Operation im Pfad "*/search-doi/*"verantwortlich ist. Der Pfad stellt eine Ressource dar, dessen Inhalt man abfragen möchte. Der Abruf der Ressource erfolgt mit der Eingabe einer gültigen DOI. Die GET-Operation "doiRequests(...)" erstellt mit der gegebenen DOI einen selbstdefinierten AsyncRequest-Objekt, mit der Anfragen an die Endpunkte der verschiedenen Dienstanbieter gesendet wird, um die Metadaten von Publikationen mit der gegbenen DOI zu erhalten. Im Anschluss wird überprüft, ob die angefragten Dienstanbieter Ergebnisse zu der gegebenen DOI geliefert haben. Falls keine Provider Metadaten liefern konnten, gibt die Schnittstelle eine Meldung aus, in der gesagt wird, dass man keine Ergebnisse zu der DOI findet. Wenn die Provider jedoch Metadaten liefern, wird das Objekt *DatasetAssessment* erstellt, um es zu bewerten und gibt am Ende den Datensatz mit den hochwertigsten Metadaten zurück.
Nachdem man solche HTTP-Operationen definiert hat, aktualisiert FastAPI automatisch die Dokumentation der gebauten REST-API (siehe Anhang Abbildung A.3)

6.1.2 OU1: API-Anfrage mit Publikationstitel

Der Abruf eines Datensatzes über den Publikationstitel ist ähnlich zu dem der DOI. Nur hat die Abfrage über den Publikationstitel die Besonderheit, dass man mit dem Titel eine Publikation nicht eindeutig identifizieren kann. Der Unterschied

liegt nicht nur bei der Bezeichnung, Pfad und Query-Parameter der GET-Operation, sondern auch im Verfahren des Metadaten-Abrufs. Aus den 3 gewählten Provider, ist Semantic Scholar der einzige Provider, der einen Abruf über den Publikationstitel ermöglicht. Der Datensatz, den Semantic Scholar liefert, beinhaltet externe IDs im Feld "*externalIds*" (siehe Beispiel-JSON 6.2). Darunter findet man auch die DOI, die extrahiert wird, um bei allen Providern die Datensätze abzurufen. Um zu diesen Schritt zu gelangen, wird die Methode "*titleGetMeta()*" des *AsyncRequest*-Objektes aufgerufen, um bei Semantic Scholar nach dem Titel zu fragen. Im Anschluss listet Semantic Scholar eine Reihe von Datensätzen zurück, die zum eingegebenen Titel gefunden werden konnten. Mit der *thefuzz*[2]-Library kann man mit der *Levenshtein-Distanz*-Formel die Ähnlichkeit von 2 Zeichenketten ermitteln. Mit dem simplen *fuzz.ratio(s1, s2)* wird der Datensatz mit der höchsten Ähnlichkeit zum angefragten Titel ermittelt, um die DOI zu extrahieren. Jedoch werden hier alle Datensätze ohne DOI ignoriert. Sobald man die DOI erhält, ist das weitere Verfahren zu dem der DOI gleich. Falls man jedoch keine DOI erhält, wird dem Nutzer mitgeteilt, dass keine Datensätze zu dem Titel gefunden wurde.


```
{
    "paperId": "b6383ae19069a0c48f39a03fad2ca00f7276815c",
    "externalIds": {
        "MAG": "2899341861",
        "DOI": "10.1016/J.TRPRO.2018.10.057",
        "CorpusId": 169570948
    },
    "url": "https://www.semanticscholar.org/paper/b6383ae19069a0c48f39a03fad2ca00f7276815c",
    "title": "Reporting quality of travel and non-travel activities: A comparison of three different survey formats",
    "abstract": null,
    "venue": "",
```


[2] https://github.com/seatgeek/thefuzz, Zugriff 17.05.2022

```
    "year": 2018,
    "fieldsOfStudy": [
        "Geography"
    ],
    "authors": [
        {
            "authorId": "2515145",
            "name": "F. Aschauer",
            "affiliations": []
        },
        {
            "authorId": "3386842",
            "name": "Reinhard Hoessinger",
            "affiliations": []
        },
        {
            "authorId": "72385262",
            "name": "B. Schmid",
            "affiliations": []
        },
        {
            "authorId": "6237165",
            "name": "R. Gerike",
            "affiliations": []
        }
    ]
}
```


Listing 6.2: Beispiel-Datensatz Semantic Scholar Publikationstitel

6.2 U3: Abruf der Provider-Ressourcen

Für die Qualitätsbewertung wird mindestens 1 Datensatz benötigt, den man von den Endpunkten der verschiedenen Provider abrufen kann. Die Endpunkte wurden in einer Config-Datei definiert und können mit der *configParser*[3]-Library ausgelesen werden. In der Config-Datei sind noch Informationen zur Abfrage der Metadaten enthalten, die bei einem GET-Request mit dabei sein muss.
Der Quell-Code 6.3 beschreibt, den Ablauf der Anfragen und die Verarbeitung der Responses.

```
async def doiGetMeta(self):
    async with aiohttp.ClientSession() as session:
        # Liste von Anfragen an Semantic Scholar
        rest_tasks = self.get_restTasks(session, self.
    get_semscholarUrl())
        # Liste von Anfragen an CrossRef
        rest_tasks2 = self.get_restTasks(session, self.
    get_crossrefUrl())
        # Liste von Anfragen an DataCite
        graphql_tasks = self.get_graphqlTasks(session)


        rest_responses = await asyncio.gather(*rest_tasks)
        rest_responses2 = await asyncio.gather(*rest_tasks2)
        graphql_responses = await asyncio.gather(*graphql_tasks)


        [self.results.append(await self.makeSem(rsp))
         for rsp in rest_responses if rsp.status == 200]
        [self.results.append(await self.makeCross(rsp2))
         for rsp2 in rest_responses2 if rsp2.status == 200]
        [self.results.append(await self.makeCite(rsp)) for rsp in
    graphql_responses if rsp.status == 200 and (await rsp.json())["
    data"]]
```

Listing 6.3: Query Ressources

Falls man mit Python noch keine nebenläufige Programme geschrieben hat,

[3] `https://docs.python.org/3/library/configparser.html` Zugriff 20.05.2022

werden die 2 Schlüsselwörter *async* und *await* unbekannt sein. Die beiden Schlüsselwörter erhält man durch den Import der *asyncio*[4]-Library und erlaubt die Nutzung asynchroner Objekte (wie der "*aiohttp.ClientSession*"), die dabei helfen die User-Story 0U2 zu erfüllen. Mit der async-await Syntax wird in der ersten Zeile die asynchrone Funktion (im englischen: Coroutine) *doiGetMeta(self)* definiert. Innerhalb des Funktionsblocks der Coroutine werden an einigen Stellen, die mit *await* gekennzeichnet sind, auf die Ergebnisse von Coroutinen gewartet, bis der Prozess im Funktionsblock weitergführt wird.

Am Anfang des Funktionsblocks wird ein asynchroner Session-Objekt über die *aiohttp*-Library definiert, mit der man asynchrone Anfragen verschicken kann, sodass man nicht auf eine Response des Providers warten muss bis man die nächste Anfrage versendet. Es werden innerhalb des Session-Blocks eine Liste von HTTP-Operationen der Provider gesammelt. In den HTTP-Operationen wird der Endpunkt, von dem man die Metadaten abrufen möchte, bekanntgegeben. In Zeile 11-13 werden die HTTP-Operationen (aus den jeweiligen Listen) gleichzeitg ausgeführt. Die Ergebnisse, die man erhält, werden in Zeile 16-20 nach dem Status-Code gefiltert, sodass man am Ende eine Liste von Datensätzen erhält, die nicht leer ist.
Man kann aber erkennen, dass DataCite anders behandelt wird als die restlichen Provider. Falls man bei DataCite über die REST-API Metadaten zu einer Publikation abrufen möchte, werden Fälle auftreten, in der DataCite keine Ergebnisse liefert obwohl man den Eintrag der Publikation über deren Suchzeile finden kann. Das liegt daran, dass diese Einträge von anderen Providern registriert sind und man über die REST-API nur die von DataCite registrierten Einträge erhält. Um aber alle Einträge aus DataCite zu erhalten (unabhängig, welcher Provider es registriert hat), bietet DataCite eine Zugriffsmethode über *GraphQL*[5]. Mit GraphQL wird nur der HTTP-Verb POST genutzt. Es bietet den Vorteil, in dem Body einer POST-Operation zu spezifizieren, welche Metadaten von Interesse sind und zurückgeliefert werden sollen. Bei der Filterung der Responses einer POST-Operation, muss neben dem Status-Code auch der Inhalt der Response

[4] `https://docs.python.org/3/library/asyncio-task.html`, Zugriff 20.05.2022

[5] `https://support.datacite.org/docs/datacite-graphql-api-guide#using-graphql-to-query-the-pid-graph`

geprüft werden. Denn der Inhalt einer Response einer Post-Operation kann leer sein, während der Status-Code den Wert 200 hält.

6.3 U1: Qualitätsbewertung & Metriken der Dimensionen

Bevor die Qualitätsbewertung durchgeführt wird, hat das Programm ein Custom-Provider-Objekt zu den einzelnen Providern und deren gelieferten Datensätze erstellt (siehe Quell-Code 6.3, Zeile 16-20). Das Custom-Provider-Objekt beinhaltet den Datensatz (im JSON-Format) selbst und alle relevanten Metadaten (DOI, Publikationstitel, Publikationsjahr, Publikationsmonat, Venue, URL und Authoren) als Instanz-Attribute sowie Funktionen, mit der man die Instanzattribute erhält. Diese Funktionen unterstützen bei der Berechnung der einzelnen Dimensionen, die in der Wrapper-Funktion *computeQuality()* der selbstdefinierten Klasse *DatasetAssessement* aufgerufen werden. Der Qualitäts-Grad setzt sich aus den einzelnen Dimensions-Grade und den Gewichtungen der Dimensionen. Die Gewichtung der einzelnen Dimensionen basiert auf deren Prioritätsstufe essentiell:=0.5, wichtig:=0.35 und nützlich :=0.15. Da alle Dimensionen essentiell eingestuft wurden sind, kann man sie ein zusätzliches Mal priorisieren.

6.3.1 Completeness

Die Completeness setzt sich aus der Division der Anzahl von vorhandenen relevanten Metadaten in einem Datensatz (Zähler) und der Anzahl aller relevanten Metadaten (Nenner) zusammen. Um die beiden Werte zu ermitteln, werden die Instanz-Attribute der Custom-Provider-Objekte genutzt. Alle Instanz-Attribute repräsentieren die relevanten Metadaten und Instanz-Attribute mit einem *None*-Wert repräsentieren relevante Metadaten, die im Datensatz des jeweiligen Providers nicht enthalten sind.
Es werden die 2 eingebauten Python-Methoden *dir()* und *getattr()* für die

Ermittlung der beiden Werte genutzt. *dir()* hilft dabei alle Attribute des Custom-Provider-Objektes aufzulisten. Da nicht alle der gelisteten Attribute berücksichtigt werden und nur die Instanz-Attribute interessant sind, kann man in Kombination mit *getattr()* den Typ und Wert des Attributs herausfinden. Indem alle Attribute des Types "*method*" ausgeschlossen werden, bleiben nur noch die Instanz-Attribute übrig, dessen Anzahl die Anzahl aller relevanten Attribute darstellt. Filtert man sie auf nicht *None*-Werte erhält man die Anzahl der vorhandenen relevanten Metadaten. Nachdem die beiden Werte ermittelt wurden sind, kann man nun dessen Division bilden.

6.3.2 Understandability

Für die Metrik in *Understandability* sind Metadatenfelder mit Authoren-Namen interessant. In einem Fall, in der ein Forscher die Arbeit eines bestimmten Authors verfolgt und dessen Nachname weit verbreitet ist (wie z.B. Nguyen, Müller, Lee, usw...), ist es nicht selten, wenn Aliase genutzt werden und der Anfangsbuchstabe der Namensteile (Nachname ausgenommen) gleich sind. Aus diesem Grund würde man für diesen Fall ausgeschriebene Namen bevorzugen, um diese besser identifizieren zu können. Ein weiteres interessantes Merkmal sind Betonungszeichen und Umlaute aus anderen Sprachen, die im Englischen nicht genutzt werden, welches die Berechnung der Metrik erschwert.
Diese Funktion 6.4 erläutert, wie der Understandability-Grad bestimmt wird:

```
1 def computeUnderstandability(self, rspObj, maxLength):
2     authors = rspObj.getAuthors() if rspObj.getAuthors() else []
3 
4     matchingName_ratio = self.determineMatchingNamePattern(authors)
5     lng_ratio = self.totalLength(
6         authors) / maxLength if maxLength > 0 else 0
7 
8     return ((matchingName_ratio + lng_ratio) / 2.0) * float(self.
    conf_obj["ASSESSMENT_WEIGHTS"]["undst"])
```

Listing 6.4: Understandability Metrik

In der Funktion *computeUnderstandability(self, ...)* werden nur 2 Aspekte

betrachtet. Im ersten Aspekt wird das Verhältnis der Namen bestimmt, die das Namen-Pattern (siehe Code-Ausschnitt 6.5) erfüllen. Der zweite Aspekt ermittelt das Verhältnis der Namen, die ausgeschrieben sind.
Das Namen-Pattern wird mit diesem *Regular-Expression* 6.5) (regex) definiert:

```
1 regex = ^[\w'\-,.]*[^_!¡?  ?¿\/\\+=@#$%^&*(){}|~<>;:[\]]*$
```

Listing 6.5: Namen-Pattern

Im ersten Block der eckigen Klammer werden alle Zeichen definiert, die in einem Namen akzeptiert werden. Im zweiten Block der eckigen Klammer sind die unerlaubten Zeichen definiert, die nicht in einem Namen auftreten dürfen.
Um das Verhältnis der ausgeschriebenen Namen zu ermitteln, werden alle Authoren-Namen zu einem String verknüpft und dessen Länge bestimmt. Der Datensatz mit den meisten ausgeschriebenen Authoren-Namen bildet den größten String und hat somit die größte Länge, das durch den Parameter *maxLength* dargestellt wird.
Je mehr ausgeschriebene Namen der Datensatz beinhaltet, desto höher ist der Grad der Understandability-Dimension.

6.3.3 Objectivity

In der Metrik der Objectivity-Dimension wird versucht einen neutralen Vergleich der Metadaten aus verschiedenen Datensätzen zu machen. Es ist zu beachten, dass die Metadaten einen Fakt repräsentieren muss. Dazu werden alle relevanten Metadaten betrachtet und nur diese rausgesucht, die in jedem Datensatz den exakt selben Wert haben müssen. Zudem sollte der Wert unabhängig von der Sprache, dieselbe Abbildung haben.
In dem Artikel[6] wird der Begriff Fakt erklärt. Der Zitat vom Komödiant Ricky Gervais[6] "*If we took every science book, and every fact, and destroyed them all, in a thousand years they'd all be back, because all the same tests would [produce] the same results.*" gibt ein Beispiel für den Begriff "*Fakt*".
In der folgenden Tabelle 6.1 werden alle Metadaten gelistet, die einen Fakt

[6] `https://channelnomics.com/2018/03/two-realities-truth-and-fact-and-theyre-not-the-same/` Zugriff 23.05.2022

darstellen und das Beispiel von Komödiant Ricky Gervais entspricht. Aus der Tabelle 6.1 wird entnommen, dass nur die 2 Metadaten *URL* und *Publisher* keinen Fakt darstellen. Geht man nach dem Zitat[6] von *Ricky Gervais*, ist es ein Fakt, dass man bis zum aktuellen Zeitpunkt über die gegebene URL das Paper finden kann. Jedoch kann es in Zukunft aus irgendeinem Grund dazu kommen, dass das Paper nicht mehr unter der URL aufzufinden ist. Gründe dafür wären z.B. Server wurde abgestellt, Service/Feature wird nicht mehr angeboten, usw... Vom Prinzip her gilt dasselbe auch für die Publisher des Papers.

Fakten	
Meta-Felder	Fact
DOI	true
Titel	true
Authoren	true
URL	false
Publisher	false
Publikationsjahr	true
Publikationsmonat	true

Tabelle 6.1: Fakten Felder

Die Metadaten (DOI, Titel, Authoren, Publikationsjahr) die ein Provider liefert wird auf Gleichheit mit den anderen Providern verglichen. Der Provider erhält von den anderen Providern eine Mitteilung über die Anzahl gleicher Werte. *Count* summiert die mitgeteilte Anzahl (:=Zähler, siehe Code-Ausschnitt 6.6). Wenn die Metadaten über die gesamte Datensatz-Menge gleich, ergibt es die maximal erreichbare Anzahl gleicher Metadaten (:=Nenner, siehe Code-Ausschnitt 6.6). Bildet man die Division zwischen den beiden Werten, erhält man den Grad der Objectivity:

```
1     count / ((len(cmpObj.getAuthors()) + 3.0) * len(cmpToArr))
```

Listing 6.6: Objectivity-Grad

Bemerkung: Für den Vergleich der Authorennamen wird die *thefuzz*-Library genutzt, mit der die Ähnlichkeit der Authorennamen bestimmt wird. Da die Authorennamen nicht in allen Provider dieselbe Darstellung haben (entweder

ausgeschrieben oder mit Aliase), aber mindestens 1 Namensteil ausgeschrieben ist (wie der Nachname), startet man mit der Suche nach einem ausgeschriebenen Namensteil. Sobald es gefunden wurde, werden die anderen Namensteile in dem zu vergleichenden Namen-String gesucht. Falls die ausgeschriebenen Namensteile nicht im Namen-String enthalten sind, wird überprüft, ob die Aliase von den Namensteile darin enthalten sind. Falls ja, erhält man einen positiven Vergleich (Gleichheit der Authorennamen), sonst negativ (Ungleichheit der Authorennamen).

6.3.4 Accuracy

In der Accuracy-Metrik werden Informationen aus dem Datensatz mit denen aus der realen Welt verglichen. Die Werte darin sollten syntaktisch und semantisch korrekt sein, d.h. es sollte einem bestimmten Datentyp und Format folgen und mit dem Stand aus der realen Welt übereinstimmen [16].
Die Wahl der Felder aus den Datensätzen ist hierfür sehr begrenzt, da viele Informationen schwer nachzuweisen sind. Die Felder Publikationsjahr, Publikationsmonat und DOI geben ein gutes Beispiel, um zu zeigen, dass diese Informationen mit dem Stand der realen Welt übereinstimmt:

1. *Publikationsmonat* - Es ist auf der ganzen Welt bekannt, dass es nur 12 Monate gibt (von Januar bis Dezember).

2. *Publikationsjahr* - Es ist jedem bekannt, was das aktuelle Jahr ist.

3. *DOI* - Es ist ein Idenfier, d.h. es gibt genau 1 DOI und man kann damit ein Objekt eindeutig identifizieren.

Bei der Berechnung dieser Metrik, wird überprüft, dass der angegebene Publikationsjahr nicht das aktuelle Jahr aus der realen Welt überschreitet. Anders ausgedrückt, die Jahreszahl aus dem Datensatz ist kleiner oder gleich der aktuellen Jahreszahl. Das selbe Prinzip wird auch beim Publikationsmonat angewendet, aber mit dem Unterschied, dass der Wert sich zwischen einem Intervall befinden muss, d.h. es ist zwischen dem 1. und 12. Monat. Für die DOI werden *regular Expressions (regex)* genutzt, um dessen Format zu überprüfen. Crossref hat in

eines deren Artikel [7] beschrieben mit welchem *regex* die DOI-Formate abgeglichen werden und wie viele dadurch ermittelt wurden. Das bezieht sich aber nur auf die DOI, die man in Crossref finden kann. DataCite beschreibt in deren Dokumentation[8] die DOI-Grundlagen und daraus resultierte das *Regex*:

```
# darf nicht im Suffix enthalten: "&\'<>
regex = '10[.][0-9]{4,}(?:[.][0-9]+)*/(?:(?!["&\'<>])\S)+'
```

Listing 6.7: Regex Ressources

Mit dem Regex wird sichergestellt, dass der Prefix mit "10." beginnt und mindestens 4 Ziffern danach folgen. Mit Ausnahme einiger Zeichen (siehe Kommentar von Quellcode 6.7) kann der Suffix ein beliebiger String sein. Erfüllen die Metadaten die Bedingungen, wird deren Anzahl durch 3 (DOI, Publikationsjahr und -monat) geteilt und man erhält den Grad der Accuracy.

[7] `https://www.crossref.org/blog/dois-and-matching-regular-expressions/` Zugriff 23.05.2022

[8] `https://support.datacite.org/docs/doi-basics` Zugriff 23.05.2022

7 Evaluation

7.1 Ziele

Es werden folgende Ziele verfolgt:

1. Analysiere die Ergebnisse für den Use-Case mit Titel. Erhält man dieselbe Publikation wie die von ORKG?
2. Analysiere die Qualitätsgrade der einzelnen Provider. Sind sie besser als die vom ORKG?
3. Bewerte die Auswahl der Provider. Ist die Wahl der Provider gerechtfertigt?
4. Bewerte die Geschwindigkeit der Qualitätsbewertung der Schnittstelle. Ist die Geschwindigkeit in der Anwendung angemessen?

7.2 Testaufbau

Für die Durchführung der Tests werden Datensätze von Publikationen benötigt, die alle relevanten Felder beinhalten, die in der Qualitätsbewertung berücksichtigt werden, d.h. DOI, Publikationstitel, Authorennamen, Publikationsjahr, Publikationsmonat, URL und die Venue. Der Datensatz, der all diese Informationen beinhaltet wird über den SparQL-Endpunkt aus der Seite des *Open Research Knowledge Graphs*[1] (ORKG) ermittelt. Die Informationen werden in

[1] `https://www.orkg.org/orkg/data` Zugriff 03.06.2022

einer csv-Datei zusammengefasst und mit Hilfe der *pandas*-Python-Library die 2 Use-Cases (DOI und Publikationstitel) für die Software getestet. Es werden dafür aus dem Datensatz 200 einzigartige DOIs und Publikationstitel extrahiert, die zur Abfrage der Metadaten dienen.
Die Informationen aus dem ORKG-Dienst werden mit denen aus den verschiedenen Providern verglichen und bewertet. Um solche Vergleiche und Bewertungen machen zu können müssen einige Modifikationen durchgeführt werden, da Messwerte für den Abschnitt interessant sind und nicht der hochwertigste Datensatz. Dafür wird ein Extra-Ordner für das Testen erstellt und darin eine kopierte Version aller nötigen Softwareteile eingefügt, um einige Anpassungen für den ORKG-Datensatz zu machen.

7.3 Messungen und Analyse

Der Quell-Code für alle Messungen, die in diesem Abschnitt durchgeführt wurden sind, kann man unter dem Test-Ordner finden.

7.3.1 Datensatz-Menge

Die beiden Abbildungen [7.1, 7.2] geben an, wie viele Datensätze die jeweiligen Provider zurückliefern. Da die DOIs und Publikationstitel aus dem Datensatz des ORKG-Dienstes stammen, hat der ORKG-Dienst in beiden Abbildungen 200 Publikations-Datensätze. Ignoriert man den ORKG-Dienst erkennt man eine weitere Gemeinsamkeit aus den beiden Abbildungen: DataCite liefert deutlich weniger Datensätze zurück als die beiden anderen (CrossRef und Semantic Scholar). Im Gegensatz dazu nimmt man zwischen CrossRef {(Titel, 173), (DOI, 200)} und Semantic Scholar {(Titel, 175), (DOI, 197)} keine große Lücke wahr. Semantic Scholar ist der einzige aus den 3 Providern, der die Möglichkeit einer Abfrage über einen Publikationstitel bietet, um die DOI zu extrahieren und somit die relevanten Metadaten abzufragen, d.h. CrossRef und DataCite können in dem Fall einer Abfrage über Publikationstitel niemals mehr Datensätze zurückliefern als

Semantic Scholar.

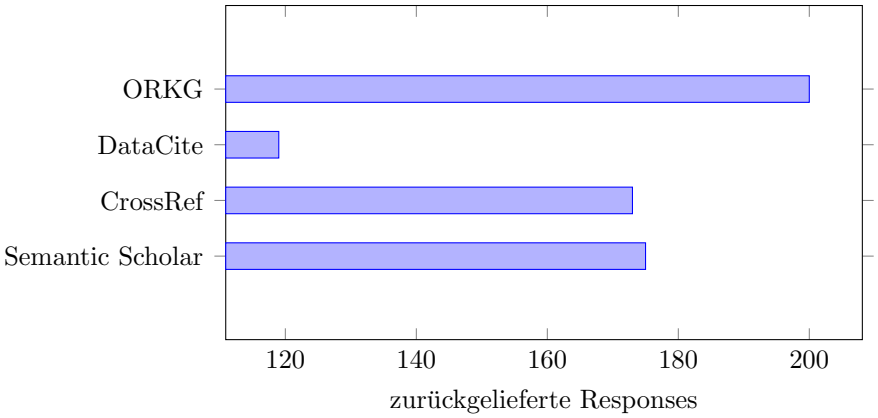

Abbildung 7.1: Anzahl zurückgelieferter Responses mit Titel

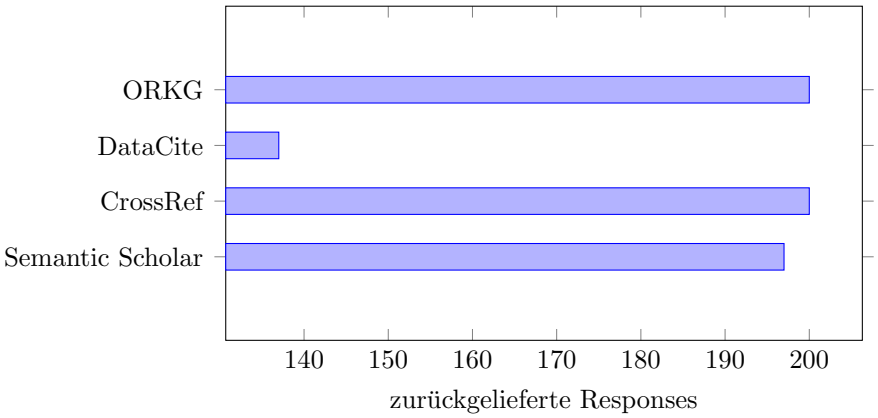

Abbildung 7.2: Anzahl zurückgelieferter Responses mit DOI

7.3.2 Dimensions- & Qualitäts-Grad

Die ersten beiden Abbildungen [7.3, 7.4] zeigen den durchschnittlichen Dimensions-Grad an und der daraus resultierende durchschnittliche Qualitätsgrad wird in der 3. Abbildung 7.5 angegeben. Wie man in Abbildung 7.5 sehen kann, liefert von den 3 Providern (ORKG-Dienst ausgeschlossen) CrossRef im Durchschnitt die hochwertigsten relevanten Metadaten. In dem Fall, in der man nach dem Titel sucht, erhält man mit CrossRef sogar einen besseren Qualitätsgrad

als der ORKG-Dienst. Es liegt auch daran, dass man mit dem Titel keine Publikation eindeutig identifizieren kann. Das führt dann dazu, dass die Metadaten der 3 Provider eine andere Publikation beschreiben als die von der ORKG. Die Dimension, die darauf einen Einfluss hat, wäre die Objectivity-Dimension (siehe Abbildung 7.3). Die restlichen Dimensionen beziehen sich nur auf den eigenen Datensatz.

Completeness: In den beiden ersten Abbildungen [7.3, 7.4] klar erkennen, dass CrossRef im Bezug auf die relevanten Metadaten mit Abstand den vollständigsten Datensatz liefert.

Understandability: Keines der Provider einen Understandbility-Grad unter 0,9 besitzt. Man kann annehmen, dass die Authorennamen zum größten Teil ausgeschrieben sind.

Accuracy: Im Falle einer Abfrage über die DOI hat CrossRef einen Accuracy-Grad von 0,995. Abgesehen von dem minimalen Unterschied, besitzen alle einen Accuracy-Grad von 1.0, d.h. die DOIs, Publikationsjahre und Publikationsmonate folgen einem Format, das in der realen Welt existiert, und diese Fakten stimmen mit der aus der realen Welt überein.

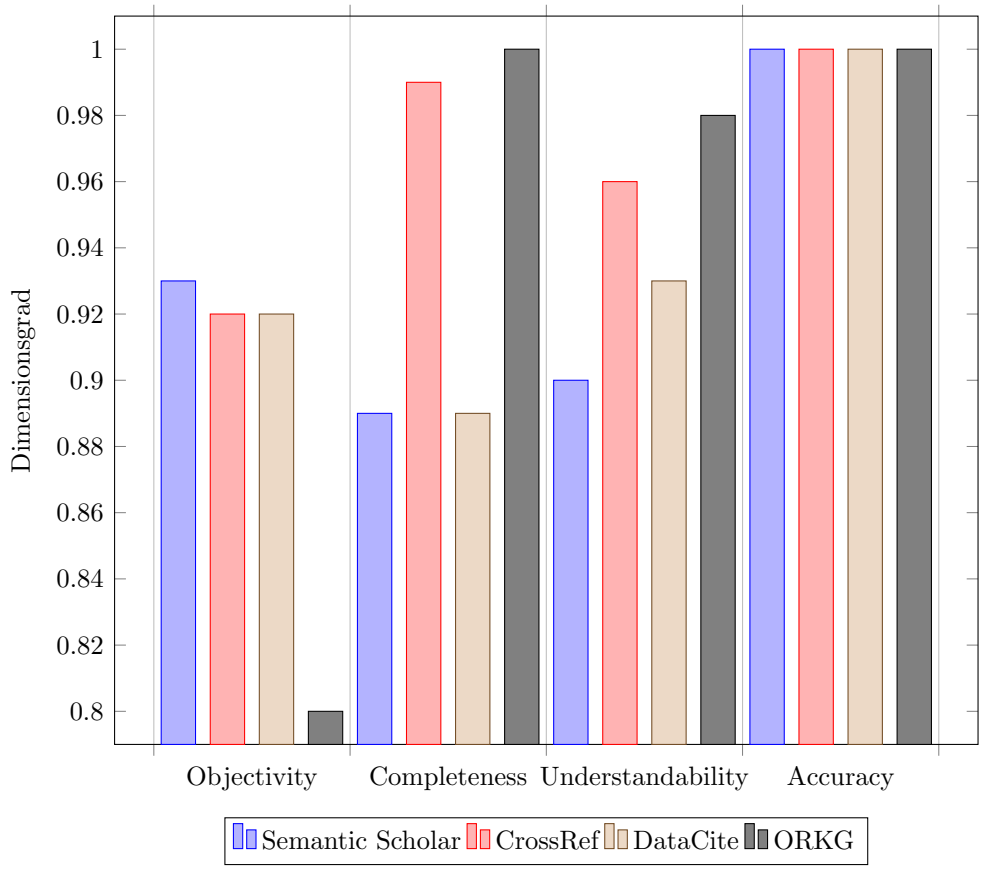

Abbildung 7.3: Dimenions-Grad der einzelnen Provider über Titel

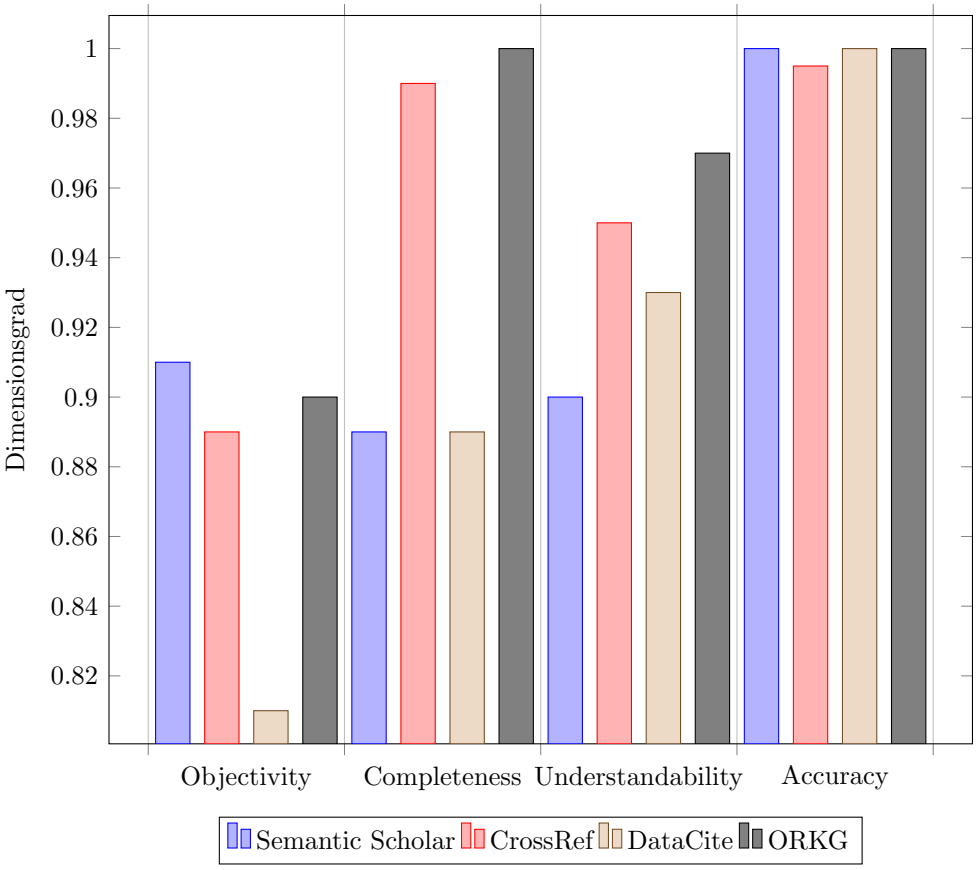

Abbildung 7.4: Dimenions-Grad der einzelnen Provider über DOI

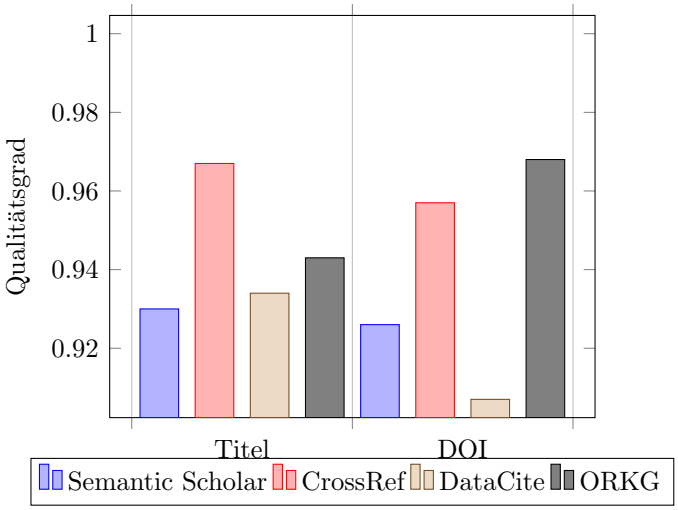

Abbildung 7.5: Qualitäts-Grad der einzelnen Provider über DOI und Titel

7.3.3 Reaktions-/Berechnungszeit

Die Abbildung 7.6 zeigt, wie schnell die Schnittstelle eine Anfrage im Durchschnitt bearbeiten kann. Man erkennt, dass die durchschnittliche Berechnungszeit bei einer Abfrage über Publikationstitel deutlich länger (ungefähr 40%) ist als eine Abfrage über die DOI. Der Grund dafür ist, dass zuerst ein Zwischenschritt gemacht werden muss bevor die "richtigen" Anfragen an die Provider verschickt werden. Den Zwischenschritt kann man dem Abschnitt 6.1.2 entnehmen. Darum sollte man, wenn möglich, die Abfrage mit Publikationstitel nur machen, wenn man selbst keine DOI zur Hand hat.

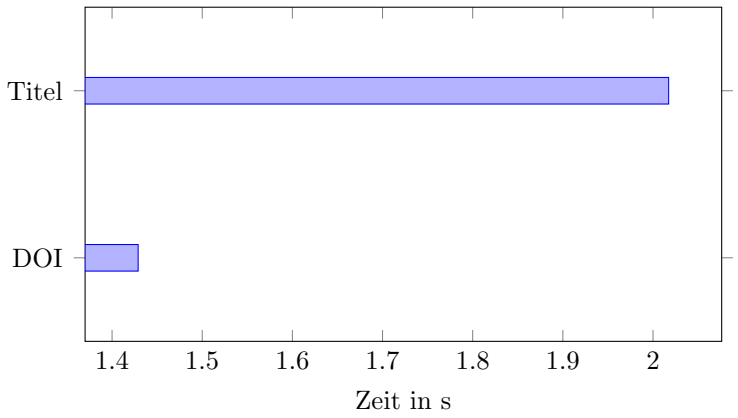

Abbildung 7.6: Durchschnittliche Berechnungszeit der Qualitätsbewertung

7.3.4 Präzision der Titel

Es wurden von 200 Titel 33 falsche oder nicht vorhandene Ergebnisse gefunden. Aus den 33 Titel hat man 4 mit demselben Titel, aber unterschiedliche DOI, und 2 mit unterschiedlichem Titel und DOI gefunden. Die restlichen 27 Titel konnten nicht gefunden werden.

8 Schlussbetrachtung

In diesem Kapitel wird ein Fazit zur These verfasst. Außerdem wird ein Ausblick für zukünftige Arbeiten bereitgestellt, um die Qualitätsbewertung der Metadaten und die gebaute REST-API zu verbessern.

8.1 Fazit

Das Ziel der These ist es, eine Schnittstelle zu bauen, mit der Dienste (wie der *Open-Research-Knowledge-Graph*), die von Metadaten abhängig sind, qualitativ hochwertige Metadaten erhalten können. Um dieses Ziel zu erreichen, wurde eine REST-API mit dem Web-Framework *FastAPI* gebaut, die von 3 Dienstanbietern die Metadaten abruft. Diese 3 Anbieter stammen aus einer Menge von 10 Dienstanbieter für Literaturrecherche, die miteinander verglichen wurden. Es wurde für die Qualitätsbewertung verschiedene Dimensionen aus verwandten Arbeiten betrachtet und priorisiert. Zu 5 Qualitätsdimensionen wurde jeweils eine Metrik definiert und für die gebaute REST-API implementiert.
Um die Funktionalität und Performance der gebauten REST-API zu validieren, wurden 200 Test-Anfragen pro Use-Case gemacht. Es wurden dabei die Grade der einzelnen Dimensionen und der Qualität der Metadaten von den Publikationen gemessen sowie die Berechnungszeit der gebauten REST-API. Die Werte der Egebnisse deuteten auf eine hohe Ähnlichkeit zwischen dem Qualitätsgrad den von der Schnittstelle gelieferten Metadaten und den Metadaten aus dem ORKG-Dienst. Jedoch wurden aus den 200 Abfragen über den Publikationstitel 173 gefunden und 6 davon lieferten Ergebnisse, die nicht von der ORKG erwartet wird (Indikator unterschiedliche DOI oder Publikationstitel). Im Vergleich zu einer

Abfrage mit DOI weiste die gebaute REST-API eine rund 40% langsamere Berechnungszeit bei einer Abfrage mit Publikationstitel auf.
Aus den Tests und Messungen wird geschlossen, dass die Funktionalität der gebauten REST-API gegeben ist und der Qualitätsgrad der Metadaten für die Anwendung des ORKG-Dienstes geeignet ist, d.h. die Wahl der Dienstanbieter ist somit gerechtfertigt. Außerdem sollten die Abfragen mit DOI priorisiert werden, falls eine hohe Abfrage-Anzahl und Präzision des angefragten Ergebnisses gewünscht ist.

8.2 Ausblick für zukünftige Arbeiten

Der Zweck der gebauten REST-API ist, dass Dienste wie der ORKG die Schnittstelle nutzt, um die qualitativ hochwertigsten Metadaten zu einer Publikation zu erhalten. In der aktuellen Version kann man die gebaute REST-API nur lokal laufen lassen, d.h. die Nutzung der Schnittstelle ist auf den lokalen Nutzer beschränkt. Um solche Dienste die Nutzung der gebauten REST-API zu ermöglichen, sollte es über einen Remote-Server laufen. Sobald es den verschiedenen Diensten möglich ist, über die gebaute REST-API hochwertige Metadaten abzurufen, gibt es mehrere Verbesserungsmöglichkeiten.
Die Qualitätsbewertung ist nur auf den ORKG-Dienst zugeschnitten. Man könnte nach verschiedenen Diensten suchen, die auf Metadaten von wissenschaftlichen Publikationen angewiesen sind. Die Dienste können mehr Metadaten (wie Abstract, Änderungsdatum, Affiliation/Zugehörigkeit) benötigen als der ORKG-Dienst und somit können einige weitere Dimensionen bei der Qualitätsbewertung berücksichtigt werden oder vorhandene Dimensionen wegfallen. Das Ziel, dass man damit verfolgt, ist eine einheitliche Qualitätsbewertung von gesamten Datensätzen, um eine Vielzahl von Diensten zu decken, die Metadaten von Publikationen benötigen.
Die nächste Verbesserungsmöglichkeit der API besteht in der Erweiterung der Use-Cases. Als Beispiel wäre die Relevanz eines Datensatzes. Falls man eine Menge von Publikationen hat und auf die Relevanz dessen Inhalts prüfen möchte, könnte man z.B. die DOIs der Publikationen nutzen, um die Metadaten zu erhalten und

falls ein Abstract enthalten ist, kann man darin nach Schlüsselwörter suchen, die auf den Inhalt der Publikation hinweisen.
In der These bezog sich die Understandability-Dimension nur auf die Authoren-Namen. Es ist möglich, dass es bessere Indikatoren für die Dimension gibt, d.h. man kann entweder den vorhandenen Indikator ersetzen oder einen weiteren Indikator hinzufügen, um eine genauere Bewertung der Dimension zu machen. Eine weitere Möglichkeit wären mehrere Metriken für eine Dimension zu definieren und implementieren. In Zaveris[16] Arbeit hat man die Metriken in subjektiv und objektiv eingeteilt. Also könnte man für die Dimensionen jeweils eine subjektive und objektive Metrik definieren.

A Anhang

Der Anhang kann Teile der Arbeit enthalten, die im Hauptteil zu weit führen würden, aber trotzdem für manche Leser interessant sein könnten. Das können z. B. die Ergebnisse weiterer Messungen sein, die im Hauptteil nicht betrachtet werden aber trotzdem durchgeführt wurden. Es ist ebenfalls möglich längere Codeabschnitte anzuhängen. Jedoch sollte der Anhang kein Ersatz für ein Repository sein und nicht einfach den gesamten Code enthalten.

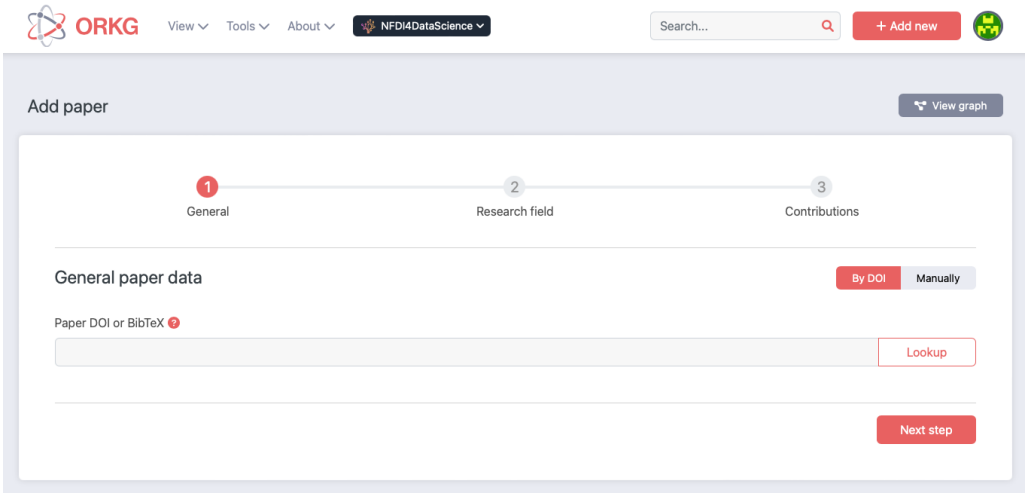

Abbildung A.1: Relevante Meta-Felder einer wissenschaftlichen Arbeit.

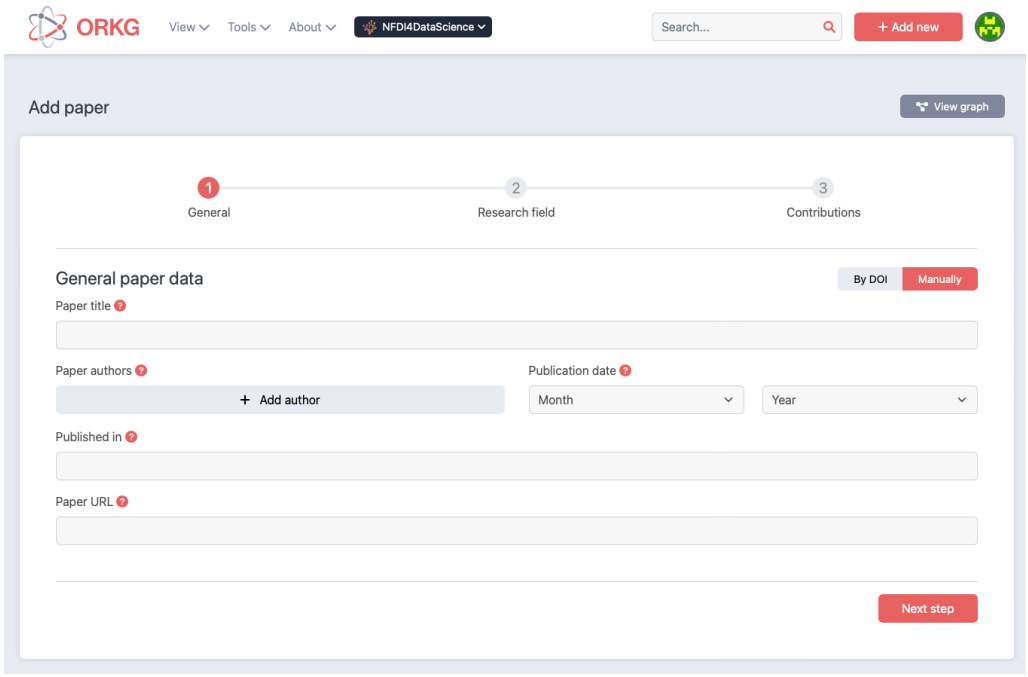

Abbildung A.2: Relevante Meta-Felder einer wissenschaftlichen Arbeit.

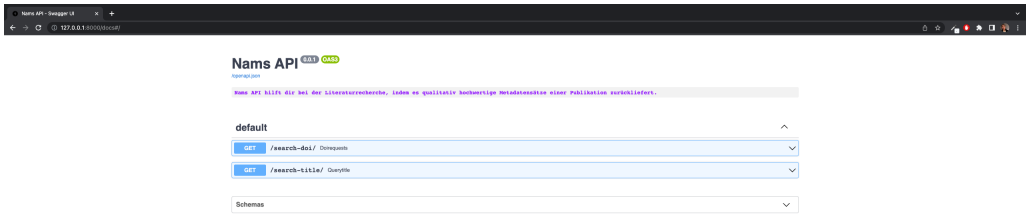

Abbildung A.3: REST-Dokumentation

B Abbildungsverzeichnis

C Tabellenverzeichnis

D Listings

E Literaturverzeichnis

[1] M. Baca, *Introduction to metadata*. Getty Publications, 2016.

[2] R. Gartner, *Metadata*. Springer.

[3] J. Pomerantz, *Metadata*. MIT Press, 2015.

[4] D. Haynes, *Metadata for Information Management and Retrieval: Understanding metadata and its use*. Facet Publishing, 2018.

[5] J. Voß, "Was sind eigentlich daten?," *LIBREAS. Library Ideas*, no. 23, pp. 4–11, 2013.

[6] F. Bodendorf, *Daten-und Wissensmanagement*. Springer-Verlag, 2013.

[7] S. Hein, "Metadaten für die langzeitarchivierung," *Standards und disziplinspezifische Lösungen*, p. 87, 2012.

[8] J. Qin and M. Zeng, *Metadata*. ALA-Neal Schuman, 2016.

[9] S. Rühle, "Kleines handbuch metadaten," in *OJ http://www. kim-forum. org/Subsites/kim/Shared Docs/Downloads/DE/Handbuch/metadaten. pdf*, 2012.

[10] M. Masse, *REST API design rulebook: designing consistent RESTful web service interfaces*. Ö'Reilly Media, Inc.", 2011.

[11] R. T. Fielding, R. N. Taylor, J. R. Erenkrantz, M. M. Gorlick, J. Whitehead, R. Khare, and P. Oreizy, "Reflections on the rest architectural style and"principled design of the modern web architecture"(impact paper award),"

in *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, pp. 4–14, 2017.

[12] F. Doglio, *Pro REST API Development with Node. js*. Apress, 2015.

[13] C. M. Kulkarni and M. Takalikar, "Analysis of rest api implementation," *Int. J. Sci. Res. Comput. Sci*, vol. 3, pp. 2456–3307, 2018.

[14] P. Király, *Measuring metadata quality*. PhD thesis, Georg-August-Universität Göttingen, 2019.

[15] R. F. D. M. M. W. Group *et al.*, "Fair data maturity model: specification and guidelines," *Research Data Alliance. DOI*, vol. 10, 2020.

[16] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer, "Quality assessment for linked data: A survey," *Semantic Web*, vol. 7, no. 1, pp. 63–93, 2016.

[17] C. Bizer, *Quality-driven information filtering in the context of web-based information systems*. PhD thesis, 2007.

[18] K. Hildebrand and M. Gebauer, *Daten-und Informationsqualität*. Springer.

[19] C. Fürber and M. Hepp, "Swiqa–a semantic web information quality assessment framework," 2011.

[20] L. L. Pipino, Y. W. Lee, and R. Y. Wang, "Data quality assessment," *Communications of the ACM*, vol. 45, no. 4, pp. 211–218, 2002.

[21] P. N. Mendes, H. Mühleisen, and C. Bizer, "Sieve: linked data quality assessment and fusion," in *Proceedings of the 2012 joint EDBT/ICDT workshops*, pp. 116–123, 2012.

[22] A. Flemming, "Quality characteristics of linked data publishing datasources," *Master's thesis, Humboldt-Universität of Berlin*, 2010.

Anerkennung

Ich möchte mich noch recht herzlich bei folgenden Personen bedanken, die mir bei der These beigestanden haben.
Dr. rer. nat. Oliver Karras für die Bereitstellung der Motivation und Feedbacks.
Prof. Dr. Sören Auer für die Untersuchung der These.
Meine Freunde und Familie, die mich auf dem Weg ermutigt und motiviert haben.

Inhalte der CD

Dieser Bachelorarbeit ist eine CD beigelegt, die alle entstandenen Ergebnisse beinhaltet:

1. Bachelorarbeit in digitaler Form (PDF-Datei)
2. Die entwickelte REST-API
3. Evaluationsunterlagen bestehend aus:
    a) Programm, dass die Tests und Messungen durchführt
    b) Messungsergebnisse