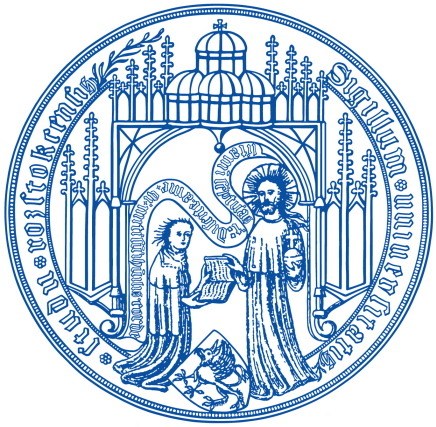
Heraldik und Semantic Web: Redesign einer Wappendatenbank

Bachelorarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik



vorgelegt von: Joris Thiele
Matrikelnummer: 217203292
geboren am: 25. März 1999 in Stendal
Erstgutachter: Dr. Holger Meyer
Zweitgutachter: Prof. Dr. Andreas Heuer
Betreuer: Dr. Holger Meyer
Michael Voß
Abgabedatum: 29. August 2021

Zusammenfassung

Die wachsende Bedeutung des Forschungsfelds der Digital Humanities als digitale Kultur- und Geisteswissenschaften in den vergangenen Jahren ist unbestreitbar. Ein Feld, dem dabei bisher vergleichsweise wenig Beachtung geschenkt wurde, ist die Heraldik oder Wappenkunde als Hilfswissenschaft der Geschichtsforschung. Bis heute existiert kein gängiger Standard zur Speicherung von Wappen in digitaler Form. Für einen solchen spielen zwei Aspekte eine wichtige Rolle: die Möglichkeit der Recherche nach dem Inhalt des Wappens und die Formalisierung der der Heraldik eigenen Blasonierungssprache. In dieser Bachelorarbeit wird eine Idee zur Wappenbeschreibung vorgestellt, die beide genannten Punkte ermöglicht, die sogenannten *Begriffsketten*. In einer prototypischen Webanwendung wird ein System implementiert, das mit diesen Begriffsketten arbeitet und die Erfassung von Wappen und die Recherche danach ermöglicht.

The increasing popularity in the field of digital humanities in recent years cannot be denied. Heraldry as an auxiliary science for historical research has not been paid much attention though. To this date, no common standard has been established for digitally storing coat of arms. For such a standard two aspects play an important role: First, search for the actual content of a coat of arms has to be possible. Second, the blazon language has to be formalised in some way. This bachelor thesis proposes an idea which satisfies both conditions, the so called *Term Chains* (from German „Begriffsketten“). A prototypical web application implements a system that works with these Term Chains and allows the collection of coat of arms and search for them.

Inhaltsverzeichnis

1 Einleitung **7**
- 1.1 Motivation . . . 7
- 1.2 Zielsetzung der Arbeit . . . 9
- 1.3 Aufbau der Arbeit . . . 9

2 Grundlagen **11**
- 2.1 Heraldik . . . 11
  - 2.1.1 Struktur von Wappen . . . 11
  - 2.1.2 Die Blasonierung . . . 14
- 2.2 Semantic Web . . . 15
- 2.3 Das Property-Graph-Modell . . . 16

3 Stand der Forschung **19**
- 3.1 Bestehende Wappendatenbanken . . . 20
  - 3.1.1 Das HERON-Projekt . . . 20
  - 3.1.2 Ordinary of Medieval Armorials . . . 21
- 3.2 Heraldik und Semantic Web . . . 22
- 3.3 Die Wappendatenbank von Michael Voß . . . 25

4 Konzept **29**
- 4.1 Das Rollenmodell . . . 29
- 4.2 Aktuelle Anwendung . . . 30
- 4.3 Neue Anwendung . . . 32
  - 4.3.1 Funktionsumfang der neuen Anwendung . . . 32
  - 4.3.2 Nicht umgesetzte Funktionalitäten . . . 35

5 Implementierung **37**
- 5.1 Verwendete Technologien . . . 37
  - 5.1.1 Neo4j . . . 37
  - 5.1.2 Flask . . . 40
  - 5.1.3 Angular . . . 40
- 5.2 Entwicklung der neuen Anwendung . . . 43
  - 5.2.1 Neo4j-Datenbank . . . 43
  - 5.2.2 Flask-API . . . 43
  - 5.2.3 Angular-Frontend . . . 51

6 Zusammenfassung und Ausblick **59**

Literaturverzeichnis **61**

Abbildungsverzeichnis

2.1 Bestandteile von Wappen ([BH17], S. 16) . . . 12
2.2 Heraldische Farben und Pelzwerke ([BH17], S. 88) . . . 13
2.3 Einige Aufteilungen von Wappenschilden ([BH17], S. 101f) . . . 13
2.4 Verschiedene Helmformen über die Jahrhunderte ([BH17], S. 129) . . . 14
2.5 Wappen von Mecklenburg-Vorpommern und der Hansestadt Rostock . . . 15
2.6 GKG-Ergebnis bei Suche nach „Günther Jauch“ . . . 17
2.7 Modellierungsbeispiel mit dem Property-Graph-Modell ([Neo]) . . . 18

3.1 Blasonierung des kleinen Landeswappens von Mecklenburg-Vorpommern nach dem in [HR20] vorgeschlagenen Schema . . . 24
3.2 Ausschnitt aus einem stark vereinfachten Begriffsgraphen . . . 27

4.1 Screenshot der Ansicht im Begriffseditor (bestehende Anwendung) . . . 31
4.2 Screenshot der Wappenerfassung (bestehende Anwendung) . . . 31
4.3 ER-Diagramm des Datenmodells für die neue Anwendung . . . 33

5.1 Screenshot des Neo4j-Browsers mit dem Begriffsgraphen . . . 38
5.2 Cypher: Möglichkeiten zur Definition von Knoten . . . 38
5.3 Cypher: Möglichkeiten zur Definition von Relationships . . . 39
5.4 Cypher: Angabe von Properties für Nodes und Relationships . . . 39
5.5 Cypher: Beispiel für eine Anfrage . . . 39
5.6 Cypher: Beispiele für `CREATE`-, `SET`-, `REMOVE`- und `DELETE`-Klauseln . . . 40
5.7 Flask: Minimalbeispiel für eine REST-API mit einem Endpunkt . . . 41
5.8 Angular: Minimalbeispiel einer Component-Klasse . . . 41
5.9 Angular: Beispiele für modifizierte HTML-Syntax . . . 42
5.10 Veranschaulichung der Datenbank im Neo4j-Browser . . . 44
5.11 Flask-API: Funktionen zur Serialisierung von Wappen, Begriffsketten und Begriffen 45
5.12 Frontend: Übersicht über die verfügbaren URLs mit den zugehörigen Komponenten 52
5.13 Frontend: Ansicht des ausgeklappten Sidenav-Elements . . . 53
5.14 Frontend: Beispiel für eine Service-Klasse . . . 53
5.15 Frontend: Abgrenzungen der Komponenten im Begriffseditor . . . 54
5.16 Frontend: Vereinfachtes DOM der Begriffseditor-Komponente . . . 55
5.17 Frontend: Screenshot der Übersicht über alle Wappen . . . 56
5.18 Frontend: Ansicht zur Bearbeitung eines Wappens . . . 56
5.19 Frontend: Wappenrecherche . . . 57

Kapitel 1

Einleitung

Seit der Nutzung des Internets durch breite Bevölkerungsschichten hat sich *Crowdsourcing* als eine Möglichkeit etabliert, vergleichsweise simple Aufgaben auf Freiwillige außerhalb der eigenen Organisation auszulagern. Das prominenteste Beispiel hierfür ist vermutlich die *Wikipedia*, in der bisher fast 180 Millionen Artikel als Online-Enzyklopädie zusammengetragen wurden [1]. Daneben haben auch speziell kulturelle Institutionen wie Museen oder Bibliotheken für sich die Möglichkeiten entdeckt, die das Crowdsourcing bietet. Beispielsweise hat die *National Library of Australia* in der Vergangenheit ein Projekt zur Digitalisierung von historischen australischen Zeitungen verwaltet, bei dem die Mitarbeit von Freiwilligen ein zentraler Baustein war (Vgl. [Hol10]).
Allgemein lässt sich dieses Beispiel dem Feld der *Digital Humanities* zuordnen. Im weitesten Sinne versteht man darunter die Verwendung von Computern beziehungsweise computergestützten Methoden im Bereich der Geistes- und Kulturwissenschaften. Als erster Schritt ist dazu die Digitalisierung und Aufbereitung bisher nur analog vorliegender Quellen wie Texten, Bildern oder archäologischen Fundstücken nötig. Darauf aufbauend können Forscher dann weitergehende Fragestellungen formulieren und lösen. Neben offensichtlichen Anwendungen wie beispielsweise digitalen Galerie- oder Museumsrundgängen sind aus diesen Bestrebungen auch diverse Initiativen zur Etablierung von Standards für die Kodierung und den Austausch von Daten hervorgegangen wie zum Beispiel die *Text Encoding Initiative*[2] zur digitalen Repräsentation von Texten aller Art, die *Music Encoding Initiative*[3] zur Repräsentation von musikalischen Dokumenten oder das *CIDOC Conceptual Reference Model*[4] als Framework für Ontologien in verschiedenen Domänen (Vgl. [Jan16]).
Ein Bereich, dem dabei in der Vergangenheit eher wenig Beachtung geschenkt wurde, ist die *Heraldik* oder Wappenkunde als Hilfswissenschaft der Geschichtsforschung.

1.1 Motivation

Obwohl Wappen in mittelalterlichen Quellen allgegenwärtig sind, gibt es zum heutigen Zeitpunkt keinen einheitlichen Standard, diese digital abzuspeichern beziehungsweise die besondere Form der Wappenbeschreibung, die *Blasonierung* (Vgl. Kapitel 2.1.2), zum Beispiel mithilfe einer Ontologie zu formalisieren, um sie auch für Maschinen verständlich zu machen. Zwar

[1] `https://stats.wikimedia.org/#/all-wikipedia-projects/content/pages-to-date/normal|line|1-month|~total|daily`, Zahl vom 31. März 2021, zuletzt abgerufen am 27.04.2021
[2] `https://tei-c.org/`, zuletzt abgerufen am 06.05.2021
[3] `https://music-encoding.org/`, zuletzt abgerufen am 06.05.2021
[4] `http://www.cidoc-crm.org/`, zuletzt abgerufen am 06.05.2021

gab es Bestrebungen Einzelner, Wappen in Datenbanken zu sammeln und zugänglich zu machen, allerdings ist die Anzahl der Mitwirkenden in diesen Projekten überschaubar und wird der Vielzahl an Wappen aus verschiedenen Quellen nicht gerecht, sodass immer nur ein kleiner Bruchteil der Gesamtheit der Wappen erfasst wurde. Die Datenbanken fokussierten sich vorrangig auf die Sammlung von Wappen mit ihren Blasonierungen in Volltext, was eine gezielte Recherche erschwert, wenn nur bestimmte Teile des Wappens vorliegen. Ebenso wird die Verknüpfung mit Informationen aus anderen Quellen in einer umfassenden Ontologie, die für die Geschichtsforschung potenziell interessant wäre, nicht berücksichtigt.
Einen Ansatz, der diese Probleme aufgreift, liefert das Konzept der *Begriffsketten* aus der Wappendatenbank von Michael Voß:
Als Diplom-Metallgestalter und Gürtlermeister arbeitete Michael Voß nach der Wiedervereinigung bis 2012 an der Inventarisierung des Kunst- und Kulturgutes der Mecklenburgischen Landeskirche. Schnell erkannte er den Vorteil, den die Nutzung von Computern und Datenbanken gegenüber der klassischen Inventarisierung auf Papier-Formblättern brachte. Daraus ergaben sich allerdings auch zwei grundlegende Probleme:

1. Da verschiedene Leute mit der Inventarisierung betraut waren und die Objekte üblicherweise im Fließtext beschrieben wurden, war nicht sichergestellt, dass für dieselbe Aussage immer dasselbe Vokabular verwendet wurde. Bei einer späteren Recherche nach einem Gegenstand würde das gelieferte Ergebnis immer nur ein Teil des tatsächlichen Ergebnisses sein, nämlich genau der Teil, bei dem die genutzten Begriffe in Anfrage und Datenbank übereinstimmten. Die Objekte, die mit anderen Worten beschrieben wurden, auch wenn dasselbe gemeint war, würden im Ergebnis nicht auftauchen.

2. Michael Voß arbeitete als Quereinsteiger im Bereich der Inventarisierung, einer Tätigkeit, die traditionell eher Kunstwissenschaftler ausüben. Dadurch waren ihm viele Begriffe zu Beginn fremd und er musste sie sich zunächst erschließen, bevor die Inventarisierung zügig durchgeführt werden konnte.

Um diese Unzulänglichkeiten zu beheben, entwickelte Voß das Konzept der Begriffsketten (Vgl. Kapitel 3.3). Dadurch ist es bei der Inventarisierung nicht nötig, dass der Nutzer wissen muss, welches Vokabular bei der Eingabe zu verwenden ist. Stattdessen gibt das System eine Reihe von zuvor definierten möglichen Begriffen vor, aus denen der Nutzer dann die passenden auswählt. Auf diese Weise wird der Fließtext verkürzt auf eine Abfolge von Begriffen, die sogenannte Begriffskette. Aus der Aussage in Fließtext

„Auf der Fußplatte des Kelchfußes befindet sich ein graviertes Wappen."

wird so die Begriffskette

„Kelch: Fuß: Fußplatte: Gravur: Wappen"[5]

Während der Arbeit in den Mecklenburgischen Kirchen kam Michael Voß auch immer wieder in Kontakt mit den lokalen Orts- und Familienwappen. Dabei fiel ihm die einfache Übertragbarkeit seiner Idee auf die Art der Beschreibung dieser Wappen auf, die ähnliche Anforderungen stellt wie die Inventarisierung. Statt der Blasonierung in Volltext sollten die Wappen ähnlich wie die Kunstgegenstände mit Begriffsketten beschrieben werden.
Zunächst für die Kunstgegenstände, später auch für die Wappen, entstand in Zusammenarbeit von Michael Voß und Thomas Reimelt[6] ein erstes System[7] zur Erfassung von Wappen mit dem erwähnten Konzept der Begriffsketten. In der folgenden Arbeit soll nun an die bisher von Voß geleistete Arbeit in der digitalen Heraldik angeknüpft werden.

[5] Beispiel entnommen aus [Voß21]
[6] `http://dynamic-development.net/Kontakt`, zuletzt abgerufen am 10.05.2021
[7] `https://wappen.azurewebsites.net/Login.aspx`, zuletzt abgerufen am 10.05.2021

1.2 Zielsetzung der Arbeit

Im Wesentlichen gliedert sich die Aufgabe dieser Arbeit in zwei Teile. In einem ersten Schritt ist zu untersuchen, inwieweit sich die von Michael Voß vorgeschlagene Beschreibungsform der Begriffsketten für den Gebrauch in der digitalen Heraldik eignet. Dazu wird diese mit den Konzepten von zwei anderen Wappendatenbanken und einer ähnlichen Form der Wappenbeschreibung aus der Literatur verglichen. Die Gemeinsamkeiten und Unterschiede der verschiedenen Ansätze werden dabei herausgearbeitet.
In einem weiteren Schritt soll ein Prototyp für eine neue Anwendung entwickelt werden, die zur Wappenbeschreibung mit Begriffsketten dient. Die Arbeitsweise dieses Prototyps bleibt dabei ähnlich wie beim bestehenden System, allerdings soll er mit moderneren Tools und Frameworks implementiert werden, um auch eine langfristige Nutzung zu gewährleisten. Über den Umfang dieser Arbeit hinaus ist die Weiterentwicklung des Prototyps angedacht ebenso wie die letztendliche Einbindung der Anwendung ins Ortschronikenportal Mecklenburg-Vorpommern. In diesem Umfeld kann sich die neue Anwendung und die Arbeit mit Begriffsketten in der Heraldik dann bewähren und gegebenenfalls etablieren.
Bei beiden Arbeitsschritten soll gleichzeitig untersucht werden, inwiefern sich die Begriffsketten eignen, um sie mit Techniken des Semantic Web zu kombinieren. Bisher ist keine Ontologie für Wappen bekannt, die es ermöglichen würde, die Wappenbeschreibungen strukturiert oder semi-strukturiert statt im Volltext abzuspeichern. Die Begriffsketten liefern dafür jedoch einen guten Ansatz, da die Begriffe in ihrer verketteten Struktur einen Graphen aufspannen, anhand dessen eine Ontologie entworfen werden könnte.

1.3 Aufbau der Arbeit

Im folgenden Kapitel 2 wird zunächst eine Einführung in die zum Verständnis dieser Arbeit nötigen Grundlagen gegeben. Im Unterkapitel 2.1 geht es dabei um die Heraldik, während es in den Unterkapiteln 2.2 und 2.3 jeweils um das Semantic Web und das Property-Graph-Modell geht.
Kapitel 3 behandelt den aktuellen Forschungsstand zur digitalen Heraldik. Im Unterkapitel 3.1 werden zwei existierende Wappendatenbanken vorgestellt, bevor in Unterkapitel 3.2 ein Ansatz aus der Literatur genauer erläutert wird, bei dem die Idee für eine heraldische Ontologie beschrieben wird. In Unterkapitel 3.3 geht es dann schließlich um den Ansatz von Michael Voß zur Beschreibung von Wappen mit Begriffsketten.
Das Kapitel 4 beschäftigt sich mit dem Konzept zur Entwicklung des Prototyps zur Wappenerfassung. Dazu wird in Unterkapitel 4.1 das von Michael Voß vorgeschlagene Rollenmodell vorgestellt. In Unterkapitel 4.2 geht es dann um die aktuell genutzte Anwendung und welche Aspekte davon für den in Unterkapitel 4.3 beschriebenen neuen Prototypen übernommen werden können.
In Kapitel 5 wird die Implementierung des Prototyps erläutert. Zunächst werden die verwendeten Frameworks und Technologien in Unterkapitel 5.1 vorgestellt. Anschließend wird in Unterkapitel 5.2 auf die Umsetzung der verschiedenen Architekturkomponenten eingegangen.
Im letzten Kapitel 6 werden die Ergebnisse der Arbeit schließlich zusammengefasst und ein Ausblick auf mögliche sich anschließende Fragen gegeben.

Kapitel 2

Grundlagen

Zunächst sollen im folgenden Kapitel einige Grundbegriffe näher erläutert werden, die für das weitere Verständnis der Arbeit nötig sind. Als erstes wird ein kurzer Überblick über die Heraldik beziehungsweise Wappenkunde gegeben. Anschließend soll der Begriff des *Semantic Web* geklärt werden. Schließlich wird das *Property-Graph-Modell*, auf dem Graphdatenbanken wie *Neo4j* beruhen, beschrieben.

2.1 Heraldik

Die Verbreitung von Wappen begann etwa Mitte des 12. Jahrhunderts und begründet sich in der Notwendigkeit, im Kampf Freund und Feind unterscheiden zu können. Aufgrund der weitgehenden Ähnlichkeit von Rüstungen und Waffen verschiedener Kräfte waren eindeutige Identifikationsmerkmale für die Krieger entscheidend. Dazu wurden die bereits genutzten Armeebanner, beim Adel auch oft deren Siegelbilder, als Bemalung der Schilde übernommen.
Schnell entwickelten sich diese Wappen von persönlichen Zeichen der meist adligen Heerführer zu Kennzeichen für die gesamte Familie der jeweiligen Träger, die zusammen mit dem Landbesitz weitervererbt wurden. Im weiteren Verlauf breitete sich der Gebrauch von Wappen mit der Ausweitung der Siegelführung aufs Bürgertum aus, da sich heraldische Schilde im Siegelfeld durchsetzten. Ab dem 15. Jahrhundert hatte sich der Gebrauch von Wappen bei Adel, Klerus, höherem Bürgertum sowie als Kennzeichen von Körperschaften und Städten etabliert.
Mit der Zeit wandelten sich auch die Einsatzzwecke für Wappen: Dienten sie zunächst als reine Erkennungszeichen in der Schlacht, verlagerte sich der Gebrauch wegen Änderungen bei Kampftaktik und -ausrüstung ins Turnierwesen. Das Anbringen von heraldischen Elementen auf der Ausrüstung der Turnierteilnehmer ist vergleichbar mit der Nutzung von Farben bei heutiger Sportkleidung zur besseren Unterscheidung zwischen den Mannschaften sowohl für die Zuschauer als auch für die Teilnehmer selbst. Als auch das Turnierwesen an Bedeutung verlor, überdauerte die Verwendung von Wappen als rein dekorative Symbole für Familien, Institutionen, Städte und Staaten teils bis in die Gegenwart (Vgl. [BH17], S. 25 ff.).

2.1.1 Struktur von Wappen

Im Wesentlichen bestehen Wappen aus zwei Hauptbestandteilen, dem *Schild* und dem sogenannten *Oberwappen*. Letzteres besteht wiederum aus Helm, Helmdecken und Helmzier. Ein Überblick über die Anordnung der Bestandteile bietet Abbildung 2.1. Daneben gibt es noch Zubehör wie zum Beispiel Rang- und Würdezeichen, Schildhalter oder Wappensprüche. Im Folgenden sollen nun die wichtigsten Elemente etwas genauer beschrieben werden.

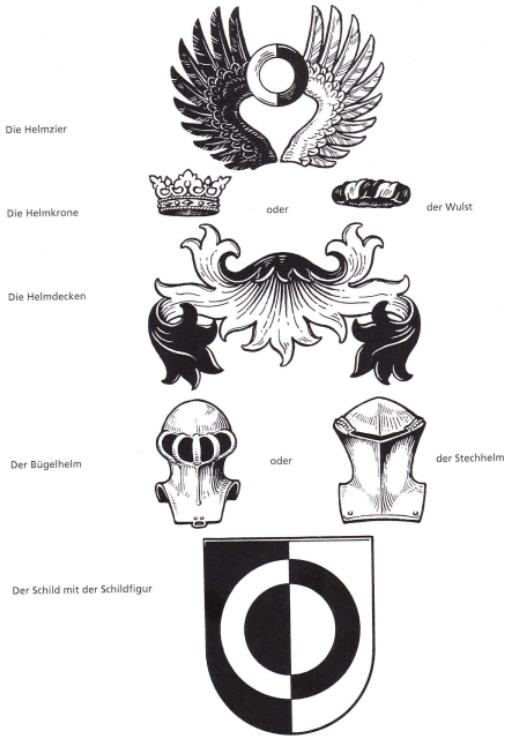


Abbildung 2.1: Bestandteile von Wappen ([BH17], S. 16)

Heraldische Farben und Pelzwerk

Die verwendeten Farben in der Heraldik beschränken sich auf Rot, Blau, Grün, Schwarz und Purpur. Daneben gibt es die beiden Metalle Gold und Silber, die auch als Gelb und Weiß dargestellt werden können. Man unterscheidet dabei nicht zwischen verschiedenen Farbtönen wie zum Beispiel hell- und dunkelgrün, sondern nutzt eine einheitliche und kräftige Grundfarbe. In Schwarzweiß-Zeichnungen werden die eigentlich farbigen Flächen als Schraffierungen dargestellt. Dabei wird beispielsweise die Farbe Rot durch eine senkrechte Schraffierung oder die Farbe Blau durch eine waagerechte Schraffierung ersetzt.
Neben den Farben werden auch sogenannte Pelzwerke genutzt, die ihren Ursprung in mit Fellen bespannten Schilden haben. Man unterscheidet Hermelin, Feh, das vom Eichhörnchen stammt, und Kürsch.
In Abbildung 2.2 findet sich eine Übersicht über die genutzten Farben und Pelzwerke.

Der Wappenschild

Der Wappenschild ist das zentrale Element jedes Wappens, der das Wappenbild zeigt. Dieses entsteht durch beliebige Kombination von geometrischen Aufteilungen und Figuren unter Verwendung der oben genannten Farben.
Bei den geometrischen Aufteilungen unterscheidet man zunächst horizontale *Teilungen* in obere und untere Hälfte (heraldisch korrekt *oben* und *unten*) von vertikalen *Spaltungen* in linke und rechte Hälfte, wobei *links* und *rechts* in der Heraldik vom imaginären Schildträger aus zu sehen sind. Diese Aufteilungen können auch miteinander kombiniert werden, um komplexere Formen mit mehreren Feldern auf dem Schild darzustellen. Ebenfalls kann die Form der

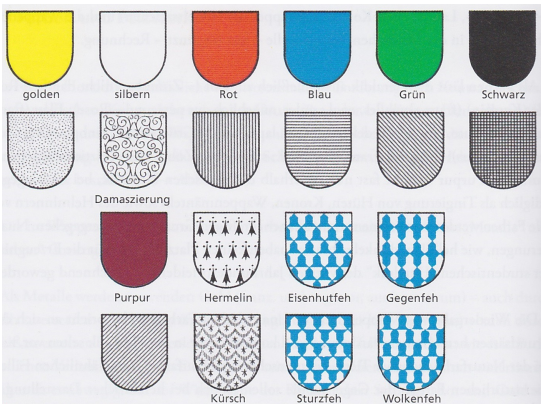


Abbildung 2.2: Heraldische Farben und Pelzwerke ([BH17], S. 88)

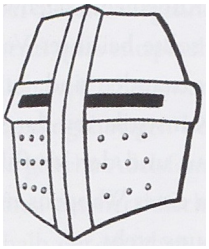
(a) Teilung (b) Spaltung (c) Halbteilung und Spaltung (d) Zinnenteilung (e) Durchgehendes Kreuz (f) Schrägteilung (g) Gestürzte Spitze

Abbildung 2.3: Einige Aufteilungen von Wappenschilden ([BH17], S. 101f)

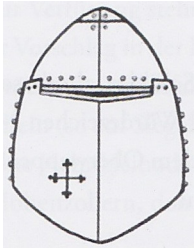
Aufteilung variieren, wenn zum Beispiel statt einer geraden Teilung eine *Zinnenteilung* vorliegt. Neben diesen grundlegenden Aufteilungen existieren auch komplexere Muster wie beispielsweise Kreuze, Schrägteilungen oder Spitzen. Eine Übersicht über die genannten Formen bietet Abbildung 2.3. Diese stellen jedoch nur einen winzigen Ausschnitt aus der Vielzahl verschiedener Ausprägungen, die auf Wappen vorkommen, dar.
Neben den geometrischen Aufteilungen sind die sogenannten *gemeinen Figuren* ein zentrales Element der Wappenbilder. Dabei handelt es sich um Abbilder von realen oder fiktiven Lebewesen oder Gegenständen aus der Umwelt des Menschen. Beispiele sind Löwen, Adler, Greifen, Blumen oder auch Himmelskörper. Auch hier ist die Vielzahl an möglichen Figuren zu groß, um an dieser Stelle einen umfassenden Überblick zu geben.

Das Oberwappen

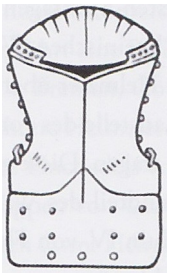
Ab Ende des 12. Jahrhunderts setzte sich vor allem in Mittel- und Nordeuropa die Verwendung des Helms oberhalb des Wappenschildes in der Heraldik durch. Die verwendeten Helmformen entsprachen weniger den tatsächlich im Kampf getragenen sondern eher den bei Turnieren verwendeten Helmen. Dennoch lässt sich eine Veränderung in der Form im Verlauf der Jahrhunderte feststellen, die in Abbildung 2.4 abgebildet ist.
Auf den Helmen wurden ab Beginn des 13. Jahrhunderts Figuren als sogenannte *Helmzier* angebracht. Ziel bei einigen Turnierkämpfen war es unter anderem, die gegnerische Helmzier abzuschlagen. Am verbreitetsten sind in der Heraldik Tierhörner und Flügel. Andere Elemente können Federn, Hüte oder auch bemalte Bretter sein.



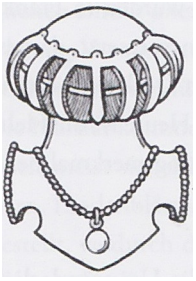
(a) Topfhelm (12. Jh)



(b) Kübelhelm (13./14. Jh)



(c) Stechhelm (14./15. Jh)



(d) Bügelhelm (16. Jh)

Abbildung 2.4: Verschiedene Helmformen über die Jahrhunderte ([BH17], S. 129)

2.1.2 Die Blasonierung

Die Blasonierung eines Wappens ist dessen Beschreibung mit einer der Heraldik eigenen Kunstsprache. Das Wort leitet sich vom französischen „blason“ („Wappen“) ab. Im Mittelalter entwickelte sich diese Kunstsprache zunächst in Frankreich und wurde später unter anderem auch an deutschen Höfen verwendet. Die damaligen Regeln sind überwiegend bis heute gültig, sodass spätmittelalterliche französische Blasonierungen auch heute noch verständlich sind. Mit dem Beginn der Renaissance ging die Praxis der Wappenbeschreibung in heraldischer Sprache zumindest im deutschsprachigen Raum verloren und wurde erst im späten 19. Jahrhundert durch heraldische Vereine wie HEROLD[1] und ADLER[2] wiederbelebt. Durch die Arbeit dieser Vereine etablierte sich ein deutschsprachiges Regelwerk zur Beschreibung von Wappen.
Der Grundsatz bei der Blasonierung liegt in einer möglichst kurzen Wappenbeschreibung. Unwichtige Details wie zum Beispiel die Form des Wappenschildes werden nicht aufgenommen. Dennoch muss die Beschreibung so präzise sein, dass ein (heraldischer) Zeichner das Wappen nur aufgrund der Blasonierung, ohne ein vorliegendes Bild, nachzeichnen kann.
Bei der Blasonierung gilt folgende Reihenfolge zur Beschreibung von Wappen:

1. Zunächst wird der Schild beschrieben beginnend mit der Farbe und geometrischen Aufteilungen. Man beginnt in der rechten oberen Ecke und beschreibt in Richtung des linken unteren Rands (die Seiten *links* und *rechts* sind in der Heraldik vom Träger des Schildes aus zu sehen).

2. Anschließend folgen die Figuren auf dem Schild und deren Stellung zueinander. Bei verschiedenen Figuren ist die wichtigste als erstes zu nennen.

3. Sollte der Schild in verschiedene Teile geteilt oder gespalten sein, wird nach der Nennung der Aufteilung jedes Feld einzeln blasoniert.

4. Nach der Beschreibung des Schildes folgt die Beschreibung des Oberwappens. Die Ausprägung des Helms wird nur genannt, falls sie vom Standard des für die betreffende Familie gültigen Standes abweicht.

Als Beispiel stehen im Folgenden die Blasonierungen des kleinen Wappens von Mecklenburg-Vorpommern (siehe Abbildung 2.5a) und des Wappens der Hansestadt Rostock (siehe Abbildung 2.5b).

[1] `https://herold-verein.de/`, zuletzt abgerufen am 22.05.2021
[2] `https://adler-wien.at/`, zuletzt abgerufen am 22.05.2021

(a) Kleines Landeswappen von Mecklenburg-Vorpommern[a]

[a] https://commons.wikimedia.org/wiki/File:Kleines_Landeswappen_Mecklenburg-Vorpommern.png, zuletzt abgerufen am 26.05.2021

(b) Wappen der Hansestadt Rostock[a]

[a] https://commons.wikimedia.org/wiki/File:Rostock_Wappen.svg, zuletzt abgerufen am 26.05.2021

Abbildung 2.5: Wappen von Mecklenburg-Vorpommern und der Hansestadt Rostock

Blasonierung des kleinen Landeswappens von Mecklenburg-Vorpommern:

„Gespalten, vorn [rechts] in Gold ein hersehender, golden gekrönter schwarzer Stierkopf mit aufgerissenem Maul, silbernen Zähnen, aufgeschlagener roter Zunge, in sieben Spitzen abgerissenem Halsfell und silbernen Hörnern, hinten [links] in Silber ein aufgerichteter, golden bewehrter roter Greif.“ ([Sch11], S. 87)

Blasonierung des Wappens der Hansestadt Rostock:
„Geteilt; oben in Blau ein schreitender, rot gezungter goldener Greif; unten geteilt von Silber und Rot.“ ([Sch11], S. 218)

2.2 Semantic Web

In ihrem 2001 in *Scientific American* veröffentlichten Artikel „The Semantic Web“[BLHL01] stellten Tim Berners-Lee und seine Koautoren James Hendler und Ora Lassila erstmals ihre Idee des Semantic Web als Erweiterung des bis dahin genutzten World Wide Web vor. Letzteres stelle dem Menschen lediglich eine breite Fülle von Informationen zur Verfügung, aus der dieser dann die für ihn in seinem Anwendungsfall relevanten heraussuchen müsse. Das Semantic Web solle es ermöglichen, dass die auf Webseiten hinterlegten Informationen, die Semantik der Seiten, von „Softwareagenten“ so durchsucht und aufbereitet werden können, dass dem Nutzer ein passendes Ergebnis zu seiner Suchanfrage gegeben werden kann, ohne dass dieser selbst die betreffenden Webseiten ansehen muss. Die Informationen im Web über menschengemachte Konzepte sollen also durch das Semantic Web auch für Maschinen verständlich sein. Berners-Lee und seine Koautoren illustrieren diese Funktionsweise an einem Beispiel, bei dem ein „Semantic Web agent“ einer Person automatisch eine Liste medizinischer Dienstleister für eine bestimmte Behandlung liefert, deren Kosten von der Versicherung der Patientin bezahlt werden und die gleichzeitig gestellte Anforderungen an Ort, Zeit und Qualität erfüllen. Ohne die semantische Funktionalität müsste der Nutzer aus einer Liste möglicher Dienstleister selbstständig diejenigen herausfinden, die die gewünschte Behandlung anbieten und die seine Versicherung übernimmt.

Pascal Hitzler[3] stellt in [Hit21] die Entwicklung des Semantic Web grob in drei Phasen dar. Zu Beginn des neuen Jahrtausends lag demnach der Fokus auf Ontologien, in denen verschiedene menschengemachte „Konzepte [...] und deren Beziehungen“ formalisiert wurden. Für den Menschen ist es beispielsweise selbstverständlich, dass ein Staat immer (mindestens) eine Hauptstadt besitzt. Für einen Computer muss diese *besitzt*-Beziehung zwischen den Konzepten *Staat* und *Hauptstadt* allerdings erst formal definiert werden, damit er damit umgehen kann. Aus diesen Bemühungen sind verschiedene W3C[4]-Standards hervorgegangen, unter anderem die *OWL* (Web Ontology Language) und das *RDF* (Resource Description Framework). Bei OWL handelt es sich um eine Sprache zur Definition von Ontologien, bei RDF um eine Darstellungsform für gerichtete, typisierte Graphen. Der Zusammenhang besteht darin, dass zunächst mit OWL eine Ontologie aus verschiedenen Typen definiert wird und anschließend in RDF konkrete Dateninstanzen gespeichert werden mit den vorher definierten Typen und Beziehungen als Knotentypen und Kanten.
Ab etwa Mitte der Nullerjahre wechselte der Schwerpunkt in der Semantic-Web-Community laut Hitzler zu *Linked (Open) Data*, wobei das „Open“ sich auf öffentliche Verfügbarkeit im Sinne von Open Source bezieht. Der Schwerpunkt in diesem Feld liegt auf der Integration von Daten aus RDF-Graphen verschiedener Domänen, zum Beispiel Geografie, Medien oder Sprachwissenschaften, in einen großen RDF-Graphen. Die Verknüpfung („Link“) der einzelnen Subgraphen geschieht dabei über URIs, die Konzepte bezeichnen, die in allen jeweils verknüpften Graphen vorkommen. Ein aus diesen Bemühungen hervorgegangenes Projekt ist *DBpedia*[5], dass auf Wikipedia-Artikeln beruht und die dahin enthaltenen Daten und Hyperlinks in RDF-Form abspeichert und öffentlich zur Verfügung stellt.
Im Jahr 2012 stellte Google den *Google Knowledge Graph*[6] (im Folgenden GKG) vor, mit dem die meisten Internetnutzer bei ihren Google-Recherchen schon in Berührung gekommen sein dürften: in kleinen Infoboxen neben den verlinkten Webseiten wird das Wissen im GKG über den gesuchten Begriff angezeigt. Abbildung 2.6 zeigt als Beispiel das Ergebnis für den Fernsehmoderator Günther Jauch. Die verlinkten Begriffe verweisen wiederum auf andere Knoten im GKG. Die Entwicklung von Wissensbasen im vergangenen Jahrzehnt wurde vor allem von den großen Internet-Konzernen getrieben und verfolgt grundsätzlich ähnliche Ziele wie zuvor bei Linked Open Data. Die Umsetzung unterscheidet sich hier jedoch bedeutend, da die Wissensbasen der Industrie nicht in ihrer Gänze öffentlich verfügbar sind, sondern nur über APIs angesprochen werden können. Weiterhin liegt die Kontrolle über das Wissen nicht bei einer großen Community mit vielen Mitwirkenden sondern in der Hand einer einzelnen Institution.

2.3 Das Property-Graph-Modell

Mittlerweile haben sich NoSQL-Datenbanken als Alternative zu den traditionellen tabellenorientierten Datenbanken mit *SQL* als Anfragesprache etabliert. Der Begriff *NoSQL* steht dabei für „Not only SQL“ ([SSH18], S. 661), deutsch „nicht ausschließlich SQL“ und umfasst eine Vielzahl von Modellen zur Datenspeicherung wie zum Beispiel Key-Value-Stores oder Document-Stores. Gemeinsames Merkmal ist dabei lediglich, dass die Modelle nicht relational sind.
Eine weitere Ausprägung von NoSQL-Datenbanken sind Graphdatenbanken, bei denen die Daten wie der Name vermuten lässt in einem Graphen gespeichert werden. Ein mögliches Modell dazu ist das Property-Graph-Modell, das zum Beispiel von *Neo4j*[7] verwendet wird und im Folgenden kurz erläutert werden soll:

[3] `http://www.pascal-hitzler.de/`, zuletzt abgerufen am 26.05.2021
[4] World Wide Web Consortium, `https://www.w3.org/`, zuletzt abgerufen am 26.05.2021
[5] `https://www.dbpedia.org/`, zuletzt abgerufen am 27.05.2021
[6] `https://developers.google.com/knowledge-graph`, zuletzt abgerufen am 27.05.2021
[7] `https://neo4j.com/`, zuletzt abgerufen am 26.05.2021

Abbildung 2.6: GKG-Ergebnis bei Suche nach „Günther Jauch“

Die abzuspeichernden Entitäten werden als *Nodes*, die Knoten des Graphen, repräsentiert. Ein Node entspricht dabei immer genau einer Entität. Die Attribute der Entität, im relationalen Modell die Spalten, werden als *Properties* des entsprechenden Nodes modelliert. Dabei handelt es sich um Schlüssel-Wert-Paare mit dem Attributnamen als Schlüssel und dem Attributwert als Wert. Die Beziehungen zwischen den Entitäten, die im relationalen Modell über Fremdschlüssel modelliert werden, sind im Property-Graph-Modell als *Relationships*, den Kanten des Graphen, dargestellt. Die Relationships sind grundsätzlich gerichtet und können wie die Nodes Attribute als Properties haben. Sowohl Nodes als auch Relationships können ein oder mehrere *Labels* haben, die die Knoten beziehungsweise Kanten Knoten- oder Kantentypen zuordnen. Die Labels fassen also Entitäten mit ähnlichen Eigenschaften zusammen und sind vergleichbar mit Tabellen im relationalen Modell.
In Abbildung 2.7 ist ein Beispiel für die Modellierung mit dem Property-Graph-Modell gegeben: beispielsweise hat der Node oben links die Properties „name“ mit dem Wert „Tom Hanks“ und „born“ mit dem Wert „1956“. Weiterhin hat der Node die Labels „Person“ und „Actor“ und steht über die „ACTED_IN“-Relationship in Beziehung zu dem „Movie“-Node darunter. Diese Relationship hat wiederum die Property „roles“ mit dem Wert „Zachry“.

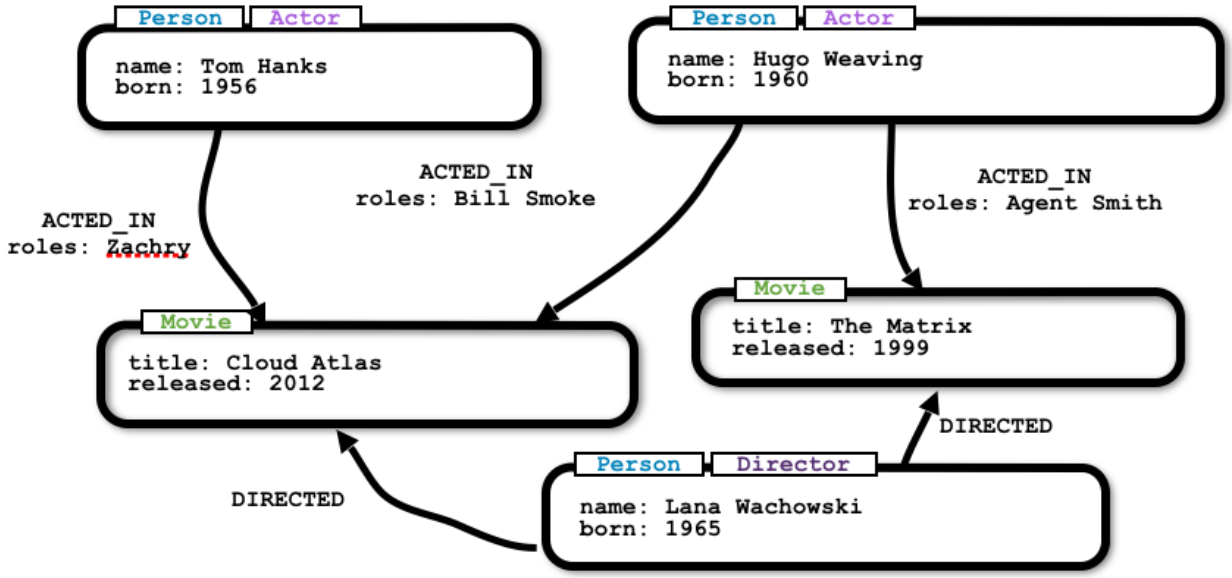


Abbildung 2.7: Modellierungsbeispiel mit dem Property-Graph-Modell ([Neo])

Kapitel 3

Stand der Forschung

Im Mittelalter und der frühen Neuzeit waren Wappen ein allgegenwärtiger Bestandteil der europäischen Kultur, deren Nutzung sich nicht nur auf die Identifikation von Personen oder Institutionen beschränkte, sondern die auch zur Darstellung von gesellschaftlichen oder politischen Zusammenhängen dienten. Dennoch spiegele sich diese Bedeutung nicht in der gegenwärtigen historischen Forschung wieder, wie Torsten Hiltmann und Thomas Riechert in [HR20] darstellen. Demnach würden Wappen bisher nur zur Identifikation der Hersteller oder Besitzer von Quellen und deren Datierung genutzt. Studierte man stattdessen intensiver den Gebrauch von Wappen in all seinen Ausprägungen, würde das eine neue Quelle zum Verständnis der mittelalterlichen Gesellschaft und Kultur eröffnen. Hiltmann und Riechert nennen eine Reihe von Gründen, weshalb die Heraldik in der Geschichtsforschung bisher vergleichsweise wenig Beachtung findet:

- Die Vielzahl: Die schiere Masse an existierenden Wappen, gedruckte Sammlungen aus dem 19. Jahrhundert umfassten bis zu 130000 verschiedene (Vgl. [UBK99]), mache es nahezu unmöglich, einen vollständigen Überblick über die Heraldik mit all seinen Facetten zu erhalten und einzelne Wappen fehlerfrei zuordnen zu können.
- Die Quellen: Wappen tauchten in verschiedensten historischen Quellen wie Büchern, Siegeln, Grabsteinen, Münzen oder Gemälden auf, da sie nicht an ein „bestimmtes Material oder Kontext“ gebunden gewesen seien. Entsprechend variierten auch die Sammelstellen von Bibliotheken über Archive bis zu Museen und anderen. Für die Befassung mit Wappen sei deshalb Kenntnis über die Details und Besonderheiten all der genannten Quellen und Sammelstellen vonnöten.
- Die Komplexität: Wie in Kapitel 2.1.2 dargestellt, wird zur Beschreibung von Wappen die Blasonierungssprache verwendet. Um die Heraldik in die Forschungsarbeit mit einzubinden, sei es demnach zunächst nötig, das hunderte Begriffe umfassende Vokabular und die Regeln der Blasonierung zu erlernen, wobei sich Begriffe teilweise auch von Sprache zu Sprache unterschieden. Die Bedeutung einiger Eigenschaften habe sich auch über die Zeit oder in verschiedenen Regionen anders entwickelt, sodass in dieser Hinsicht derselbe Begriff nicht zwangsläufig dasselbe Konzept bezeichnet. Letztendlich sei die Blasonierung „eher eine Konvention, als ein Satz von Regeln“.
- Die Ressourcen: Große Zusammenstellungen von Wappen stünden bisher vor allem in gedruckter Form zur Verfügung. Wegen des begrenzten Platzes würde das Problem der komplexen Beschreibung durch die Verwendung unklarer Abkürzungen oder mangelnder Quellenangaben noch verstärkt. Die Sammlungen seien meist nach den Namen der

> Wappenträger sortiert, was die Identifizierung eines vorliegenden unbekannten Wappens schwierig mache und lediglich Vermutungen über die Herkunft des Wappens bestätigen könne.

Die Autoren stellen schließlich die These auf, dass die Integration von heraldischem Material in die Forschungsarbeit aus genannten Gründen nur mit Methoden der Informatik möglich sei. Dazu soll im Rest dieses Kapitels zunächst ein Überblick über bereits existierende Wappendatenbanken gegeben werden. Anschließend wird der Ansatz von Torsten Hiltmann und Thomas Riechert zur Integration von Wappen ins Semantic Web näher erläutert. Als Letztes soll die Idee von Michael Voß zur Beschreibung von Wappen mit sogenannten Begriffsketten vorgestellt werden, die zentral für diese Arbeit ist.

3.1 Bestehende Wappendatenbanken

Gegenwärtig existieren bereits einige Initiativen zur Sammlung von Wappen in Datenbanken, um interessierten Nutzern einen Einstiegspunkt in die elektronische Recherche nach Wappen zu geben. Von diesen sollen im Folgenden das an der Universität Augsburg initiierte *HERON-Projekt* sowie die Datenbank *Ordinary of Medieval Armorials* des Dänen Steen Clemmensen vorgestellt werden. Weitere Ansätze werden zum Beispiel in [HR20] und [BH17] auf den Seiten 55ff. genannt.

3.1.1 Das HERON-Projekt

Das HERON-Projekt[1] lief im Zeitraum von 1997 bis 2002 an der Universität Augsburg unter Leitung von Werner Kießling. Die Zielsetzung bestand darin, mithilfe von Multimediadatenbanken Recherchewerkzeuge für Kunsthistoriker und verwandte Berufsgruppen zu realisieren, die hauptsächlich mit den Inhalten von Bildern arbeiten sollten. Als Anwendungsfeld wurde zunächst die Heraldik gewählt, da das verwendete Vokabular zur Beschreibung beziehungsweise die auftretenden Formen, Figuren und Farben vergleichsweise stark strukturiert und begrenzt ist. Dies sei nötig, um die Qualität der Ergebnisse von Bild- und herkömmlicher Volltextrecherche miteinander vergleichen zu können (Vgl. [BBUW98]). In einem weiteren Schritt hätte dann die Erweiterung auf allgemeine, weniger stark kontrollierte, Bildbestände stattfinden können.
Wie auch in [HR20] wurde die Sortierung nach Namen in klassischen gedruckten Nachschlagewerken als ein Kernproblem bei der Wappenrecherche identifiziert. Laien müssten dadurch im schlimmsten Fall das Nachschlagewerk sequentiell durchsuchen, um den Träger eines vorliegenden unbekannten Wappens zu finden. Statt der Suche nach Namen schlagen die Projektverantwortlichen die Suche nach bestimmten Bildfeatures des Wappens vor. Dabei wird der Bildinhalt der Eingabe mit den in der Datenbank gespeicherten Bildern verglichen. Der Fokus wird auf die folgenden drei Features gelegt (Vgl. [UBK99]):

- Farbfeatures: Diese werden in Farbverteilungshistogrammen repräsentiert, für deren Säulen die Farbwerte der einzelnen Pixel zusammengefasst werden. Da die Anzahl heraldischer Farben im einstelligen Bereich liegt (Vgl. Kapitel 2.1.1), sind die Histogramme wegen der beschränkten Säulenanzahl effizient vergleichbar. Aus demselben Grund sind die Farben allein allerdings auch nur bedingt aussagekräftig, da ähnliche Kombinationen häufig auftreten können. Stattdessen sollten die Farben mit anderen Features kombiniert werden.
- Texturfeatures: Diese sind vor allem beim Vergleich von Pelzwerken mit ihrem regelmäßigen Muster von Nutzen.

[1] `https://www.informatik.uni-augsburg.de/lehrstuehle/dbis/db/research/preference/finished/heron/`, zuletzt abgerufen am 25.06.2021

- Formfeatures: Dabei geht es um den Vergleich der geometrischen Formen und Figuren wie Teilungen, Spaltungen oder Kreuzen in den Wappen. Im Vergleich zu Farben und Texturen ist die Anzahl von Möglichkeiten hier ungleich größer, weswegen den Formen auch eine größere Bedeutung zukommt. Mögliche Skalierungen und Translationen werden vernachlässigt, sodass Wappen, die in anderen Wappen als Teilelemente vorkommen (beispielsweise durch Vereinigung von zwei Wappen zu einem nach einer Heirat), ins Rechercheergebnis miteinbezogen werden.

Als Datenbasis für die Bildrecherche wurden im HERON-System rund 5500 Wappen digitalisiert. Für etwa 45 Prozent wurden zusätzlich zur bildlichen Analyse auch Blasonierungen und genealogische Informationen aus den zugrunde liegenden Wappenbüchern in die Datenbank aufgenommen, was neben der Bild- auch eine Volltextsuche ermöglicht. Darüber hinaus wurde im HERON-System ein sogenannter *Bildthesaurus* geschaffen, der dem Laien den Umgang mit den der Heraldik eigenen Fachbegriffen mittels Bildbeispielen erleichtern sollte. Dieser Ansatz eignet sich auch für eine mehrsprachige Nutzung des Systems, da in verschiedenen Sprachen unterschiedlich benannte Konzepte durch dasselbe Bild dargestellt werden können (Vgl. [UBK99], [BBUW98]).
Insgesamt lassen sich im HERON-Projekt viele interessante Ansätze für die digitale Arbeit mit Wappen finden. Wie auch Hiltmann und Riechert in [HR20] kommen die Autoren zu dem Schluss, das eine Recherchemöglichkeit nach anderen Gesichtspunkten als dem bloßen Namen des Wappenträgers unabdingbar ist, damit auch Heraldik-Laien schnell ein Ergebnis zu ihren Anfragen erhalten. Ebenfalls von Bedeutung ist die Schaffung eines (potenziell mehrsprachigen) Thesaurus, um einen einfachen Einstieg in die komplexe Begriffswelt der Heraldik zu ermöglichen. Diese Idee findet sich auch im Konzept der Begriffsketten, das in Kapitel 3.3 näher erläutert wird.
Problematisch ist allerdings der nötige Aufwand, um ein vorliegendes Wappen digitalisieren zu können. Gedruckte Wappen müssen zunächst eingescannt und anschließend gegebenenfalls nachbearbeitet werden. Wappen aus anderen Quellen (Münzen, Siegel, Schilder etc.) müssen fotografiert und die Fotografien den Anforderungen entsprechend nachbearbeitet werden, um eine Arbeit mit dem System möglich zu machen. Gerade für Laien ist dieser Aufwand oftmals nicht möglich. Hierbei wäre es allerdings interessant zu sehen, inwiefern eine Weiter- oder Neuentwicklung des HERON-Systems mit dem heutigen Stand der Technik zu einer einfacheren Bedienbarkeit und besseren Rechercheergebnissen führen würde.

3.1.2 Ordinary of Medieval Armorials

Das *Ordinary of Medieval Armorials* ([Cle], im Folgenden mit *OoMA* abgekürzt) ist eine Sammlung der Inhalte aus über 500 Wappenbüchern, die erstmalig 2006 von Steen Clemmensen veröffentlicht wurde. Als studierter Biochemiker befasste sich Clemmensen in seiner Freizeit intensiv mit der Heraldik und trug die Ergebnisse seiner Arbeit im OoMA zusammen, das bisher über 87000 Wappen vor allem aus dem französisch-, englisch- und deutschsprachigen Raum umfasst. Anders als zum Beispiel beim HERON-Projekt oder in gedruckten Werken werden die Wappen nicht im Volltext blasoniert. Stattdessen hat Clemmensen ein System zur Blasonierung mit dem Fokus auf „Kürze, Klarheit und Trennung in einzelne Teile“ ([Cle], genaue URL: `http://www.armorial.dk/explanations/index.html`) erdacht, mit dem die Wappen blasoniert sind. Dabei wird die Blasonierung in zwei Teile aufgeteilt: der erste beschreibt die im Wappen vorkommenden Farben mit ihren Anfangsbuchstaben in der englischen Blasonierungssprache, der zweite weist die Aufteilungen und Figuren im Wappen aus. Auf diese Weise wird aus der Blasonierung des Kleinen Landeswappens von Mecklenburg-Vorpommern (siehe Kapitel 2.1.2) etwa

„OA S G; per pale & bull's face & griffin“,

wobei die Details des Stierkopfes und des Greifen hier zur Vereinfachung weggelassen und nur deren Hauptfarben angegeben wurden. Wie im OoMA wurde die Blasonierung auf Englisch vorgenommen. Das „OA S G“ bezeichnet die Grundfarben des Wappens, vorne Gold („Or“) und hinten Silber („Argent“), sowie die Farben des Stierkopfes, schwarz („Sable“), und des Greifen, rot („Gules“). Der Teil nach dem Semikolon bezeichnet die Teilung des Wappens („per pale“) sowie den Stierkopf („bull's face“) und den Greifen („griffin“). Die Reihenfolge der Farben und Figuren sowie Leer- und Etzeichen implizieren die Position im Wappenschild.
Die Auflistung der Wappen erfolgt nach ihrem Inhalt. Zu einer bestimmten inhaltlichen Ausprägung von Figuren und Farben finden sich also mehrere Wappen, die diese Merkmale aufweisen, mit ihrem Ursprung und dem Wappenbuch, aus dem Clemmensen sie entnommen hat. Für Nutzer ist daher eine inhaltsbasierte Suche möglich und sie müssen nicht zwangsläufig den Träger des vorliegenden Wappens kennen. Zusätzlich zu der Auflistung aller Wappen nach Inhalt gibt es spezielle Listen, in denen Clemmensen einzelne Wappenbücher ausführlich dokumentiert. Diese orientieren sich im Wesentlichen an den originalen Büchern, sind also wieder nach Namen sortiert.
Auch insgesamt ist das OoMA eher auf die Sammlung des Inhalts der Wappenbücher fokussiert, deren Ursprung teilweise im Mittelalter selbst liegt. Es stellt mehr eine Digitalisierung der verschiedenen Quellen als ein tatsächliches Recherchewerkzeug für (Laien-)Heraldiker dar: es existieren keinerlei Bilder im System, sodass Nutzer lediglich anhand der Blasonierung ein Wappen identifizieren müssten. Gerade für Laien ist das eine nahezu unmögliche Aufgabe, da das von Clemmensen entworfene System zur Blasonierung nur schwer verständlich ist und eine gewisse Einarbeitungszeit benötigt. Erschwerend kommt hinzu, dass die Inhalte verschiedener Wappenbücher sich oftmals doppeln, allerdings feine Unterschiede in der Rechtschreibung liegen. Weiterhin findet die Veröffentlichung in einer Reihe von PDF-Dateien statt, die außerhalb einer Textsuche in PDF-Readern keine inhaltlichen Anfragen erlauben.
Dennoch liefert auch das OoMA wichtige Ansätze in der Arbeit mit Wappen: einerseits sind die Wappen anhand ihres Inhalts aufgelistet, andererseits zeigt Clemmensen eine Möglichkeit der Formalisierung von Blasonierungen auf, die sich von der Volltextbeschreibung loslöst und stattdessen die wesentlichen Inhalte in standardisierter Form in den Vordergrund stellt.

3.2 Heraldik und Semantic Web

In [HR20] fassen die Autoren Hiltmann und Riechert die Unzulänglichkeiten bisheriger Ansätze zur digitalen Speicherung von Wappen wie folgt zusammen:

- Die Beschreibung von Wappen in Volltext sei fehleranfällig, solange es keine Mechanismen gebe, die Einträge auf Richtigkeit überprüften. Bereits kleinste Rechtschreibfehler in den erfassten Daten würden das Rechercheergebnis verfälschen.
- Das verwendete Vokabular in den Beschreibungen sei nirgendwo standardisiert oder spezifiziert. Begriffe könnten von verschiedenen Nutzern anders verstanden werden und zu Fehlern führen, gerade wenn das Eintragen in die Datenbank unterschiedlich abläuft.
- Mehrsprachigkeit würde von den Systemen nicht unterstützt, sodass sich der Nutzerkreis auf einzelne Länder beschränke.
- Bisherige Systeme ließen nur Anfragen an die Metadaten von Wappen und ihre heraldische Beschreibung zu. Darüber hinaus gehende Analysen, wie zum Beispiel die Häufigkeit bestimmter Eigenschaften seien nicht möglich.

Um diese Hindernisse zu überwinden, schlagen die Autoren einen Ansatz vor, der die Repräsentation von heraldischem Inhalt mit Semantic-Web-Technologien in den Mittelpunkt stellt,

speziell die Idee von Linked Data in Form von RDF und Ontologien (siehe Kapitel 2.2). Auf diese Weise würde die Wappenbeschreibung nicht mehr durch Wörter in bestimmten Sprachen stattfinden sondern auf einer konzeptionellen Ebene mithilfe von eindeutigen URIs für die einzelnen Konzepte.
Zentral dafür sei die Interpretation von Wappen als „Kombination von Konzepten“: Wie in Kapitel 2.1.1 beschrieben ist in einer bildlichen Darstellung der verwendete Farbton, beispielsweise hellrot oder dunkelrot, nicht entscheidend. Wichtig sei in diesem Fall nur, dass das heraldische Konzept *Rot* vermittelt wird. Dasselbe gelte für Figuren wie Löwen oder Greifen, bei denen Merkmale wie zum Beispiel ihre Größe nicht von Belang seien. Es müssten nur die Konzepte *Löwe* oder *Greif* erkennbar sein. Die verschiedenen Konzepte hätten dann wiederum Eigenschaften, nach denen sie differenziert werden könnten, wie beispielsweise die Farbe der Klauen oder die Körperhaltung beim Löwen.
Die Konzepte und ihre Beziehungen zueinander werden in einer Ontologie definiert, in der jedem Konzept eine eindeutige URI zugeordnet wird. Dadurch wird ein kontrolliertes Vokabular zur Wappenbeschreibung geschaffen. Das Konzept *Greif* hätte so zum Beispiel die URI **dho:Griffin**, wobei „dho“ hier und im Folgenden den Namensraum der Ontologie bezeichnet und der Bezeichnung in [HR20] entspricht. Für das Konzept kann dann der Name in verschiedenen Sprachen definiert werden, hier etwa „Greif“ in Deutsch, „Griffin“ in Englisch und „Griffon“ in Französisch. Eigenschaften werden auf dieselbe Weise beschrieben, zum Beispiel **dho:Segreant** für die Eigenschaft *Aufgerichtet* von geflügelten vierbeinigen Wappentieren. Mit diesen Bezeichnungen können sehr einfach Konzepthierarchien aufgebaut werden: beispielsweise kann **dho:Griffin** als Unterklasse von **dho:FictionalAnimals** (*Fiktive Tiere*), das als Unterklasse von **dho:Animals** (*Tiere*) und das als Unterklasse von **dho:CommonCharges** (*Gemeine Figuren*) definiert werden. Die Beziehungen zwischen den Konzepten entsprechen Prädikaten in RDF-Tripeln und können zum Beispiel als **dho:hasStyle** (für das Aussehen) oder **dho:hasTincture** (für die Farbe) dargestellt werden.
Die Speicherung der Daten, also der Wappen mit ihren Blasonierungen, erfolgt ähnlich und referenziert die zuvor definierten Begriffe der Ontologie. Die einzelnen Wappen und auch die in ihnen vorkommenden Figuren und Aufteilungen bekommen wiederum URIs, mit denen eine eindeutige Identifikation möglich ist. Der verwendete Namensraum für die Wappen (im Folgenden mit **dhd** abgekürzt) ist dabei ein anderer als der für die Konzepte der Ontologie.
Nach dieser Vorgehensweise und entsprechend definierten Konzepten würde das kleine Landeswappen Mecklenburg-Vorpommerns (siehe Abbildung 2.5a) etwa wie in Abbildung 3.1 blasoniert werden, wobei Details von Bullenkopf und Greif der Einfachheit halber weggelassen wurden.
Das beschriebene Vorgehen bietet laut Hiltmann und Riechert eine ganze Reihe an Vorteilen gegenüber klassischen gedruckten Werken und den genannten Wappendatenbanken (Vgl. [HR20]):

1. Wie bereits oben beschrieben ist Mehrsprachigkeit mit dem Ansatz sehr leicht zu implementieren. Dadurch ist der Nutzerkreis potenziell größer als bei einsprachigen Systemen und es ist wahrscheinlicher, dass ein mit dem Ansatz arbeitendes System eine kritische Masse erreicht und sich etabliert.

2. Der hierarchisch strukturierte Aufbau der Blasonierungen beziehungsweise der zugrunde liegenden Ontologie ermöglicht Anfragen nach Teilen des Wappens. Wenn von einem vorliegenden Wappen zum Beispiel nur noch die Figur, nicht aber die Farbe, erkennbar ist, kann speziell nach diesem Merkmal gesucht werden. Im Ergebnis würden dann

3. Die strukturierte und vor allem maschinenverständliche Speicherung der einzelnen Wappenbestandteile ermöglicht Analysen wie zum Beispiel Häufigkeitsverteilungen über die Ausprägung einzelner Körperteile von Wappentieren.

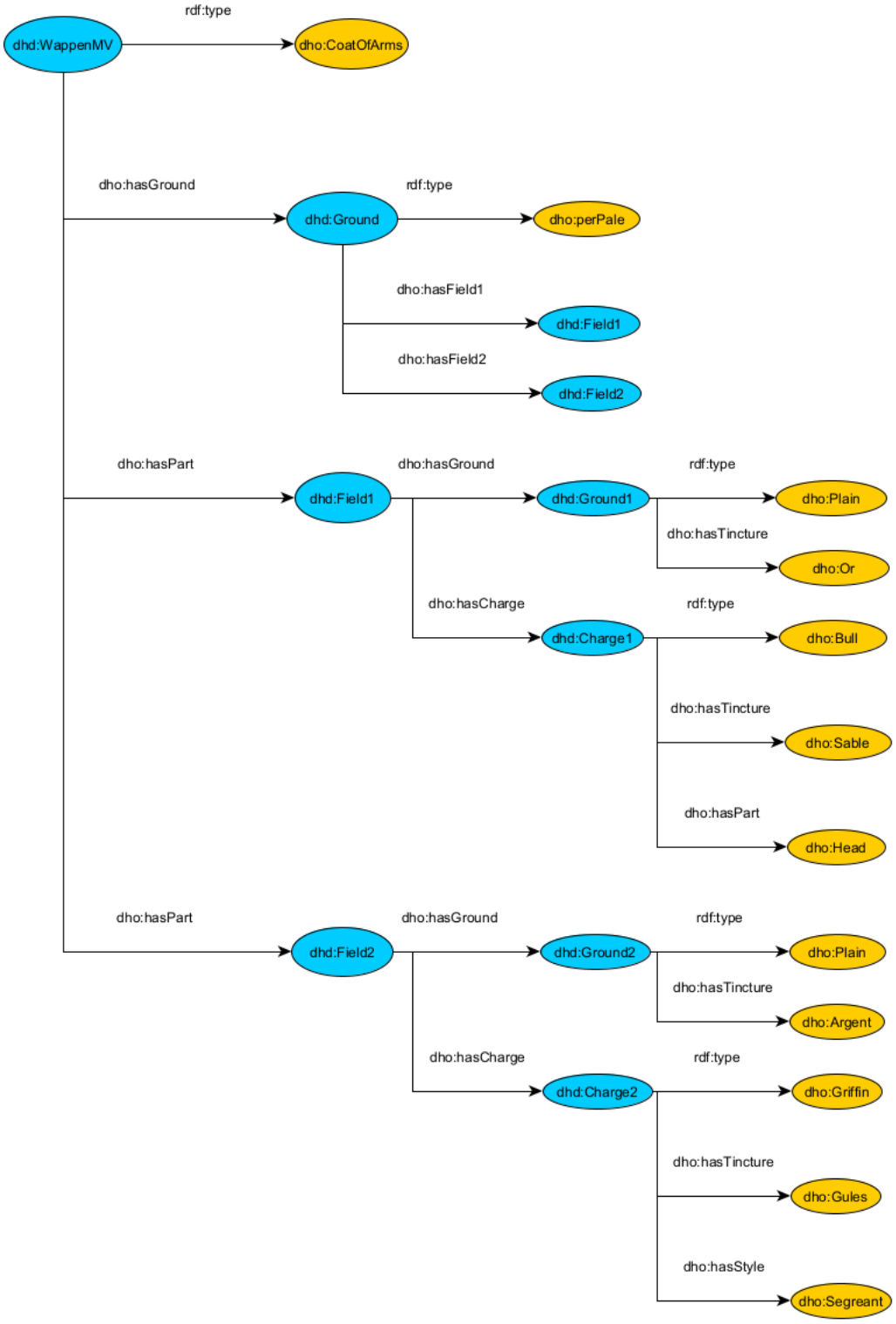


Abbildung 3.1: Blasonierung des kleinen Landeswappens von Mecklenburg-Vorpommern nach dem in [HR20] vorgeschlagenen Schema

4. Schließlich werden die Daten bei diesem Ansatz in RDF abgespeichert und sind damit in einem standardisierten und etablierten Format, für das eine ganze Reihe an Tools zur Verfügung steht. Der Austausch von Daten aus verschiedenen Quellen wäre in diesem Format vergleichsweise einfach zu gestalten.

Auf der anderen Seite steht dem eine zentrale Herausforderung gegenüber: die Grundlage des Ansatzes muss eine Ontologie sein, die das Feld der Heraldik möglichst allumfassend beschreibt, um weite Verbreitung zu erreichen. Die Ontologie muss korrekt mit heraldischen Konzepten umgehen, die in einer Kultur existieren, in anderen aber nicht. Weiterhin müssen zeitliche Veränderungen, bei denen ein Begriff im Laufe der Zeit seine Bedeutung geändert hat, korrekt aufgenommen werden.

3.3 Die Wappendatenbank von Michael Voß

Wie schon in Kapitel 1.1 kurz beschrieben entwickelte Michael Voß während seiner Arbeit für die Mecklenburgische Landeskirche die Idee der Begriffsketten, die im Folgenden noch erläutert wird (Vgl. [Voß21]). In Eigenregie ließ Voß eine Webanwendung basierend auf seiner Idee entwickeln, mit der in den folgenden Jahren etwa 12000 Objekte[2] aus den mecklenburgischen Kirchen inventarisiert wurden. Man kann hierbei also durchaus von einer etablierten Arbeits- und Vorgehensweise sprechen. Nach seinem Ruhestand erweiterte Voß das Anwendungsgebiet auf die Heraldik und erfasste bis heute[3] über 250 Adels- und Kommunalwappen vor allem aus dem mecklenburgischen Raum[4].
Für die Entwicklung der Begriffsketten spielten zwei Aspekte, die sich so später teilweise auch in [HR20] finden sollten, eine zentrale Rolle:

1. Für dieselbe Aussage sollte immer derselbe Begriff verwendet werden. Das entspricht genau der Idee aus dem vorigen Kapitel 3.2, dass Konzepte durch ein eindeutiges Merkmal identifizierbar gemacht werden. Durch die Verwendung dieses einheitlichen Vokabulars wird verhindert, dass verschiedene Nutzer unterschiedliche Begriffe für dasselbe Konzept verwenden und damit Rechercheergebnisse unvollständig werden.

2. Die Begriffswelt, sowohl bei der Inventarisierung von Kunstgut als auch in der Heraldik, ist vergleichsweise umfangreich und komplex und damit für Quereinsteiger am Anfang nur schwer zu erschließen. Eine vom System assistierte Beschreibung durch den Nutzer wäre daher von Vorteil.

Ausgehend von diesen beiden Kerngedanken leitete Voß die Begriffsketten ab, die die klassische Volltextbeschreibung eines Gegenstands auf die für die Beschreibung wesentlichen Begriffe reduziert (vergleichbar dem Stichwortverfahren beim Text Information Retrieval). Aus der Aussage

„Gespalten, vorn in Gold [...]“

wird so die Begriffskette

„Schild: gespalten: Feld 1: Farbe: Gold“[5].

[2] persönliche Mitteilung vom 01.07.2021
[3] Stand 28.05.2021
[4] `https://wappen.azurewebsites.net/Login.aspx`
[5] Die Begriffe hier und in den folgenden Beispielen wurden aus der bestehenden Wappendatenbank übernommen.

Falls mehr als eine Aussage über ein Objekt getroffen werden soll, müssen mehrere Begriffsketten angegeben werden. Die vollständige Blasonierung des kleinen Landeswappens von Mecklenburg-Vorpommern (siehe Abbildung 2.5a) sieht zum Beispiel wie folgt aus:

Schild: gespalten: Feld 1: Farbe: Gold
Schild: gespalten: Feld 1: Säugetier: Tierart: Stier
Schild: gespalten: Feld 1: Säugetier: Kopf: Farbe: schwarz
Schild: gespalten: Feld 1: Säugetier: Kopf: abgerissen
Schild: gespalten: Feld 1: Säugetier: Kopf: Maul: aufgerissen
Schild: gespalten: Feld 1: Säugetier: Kopf: Zahn: Farbe: Silber
Schild: gespalten: Feld 1: Säugetier: Kopf: Zunge: Farbe: Rot
Schild: gespalten: Feld 1: Säugetier: Kopf: Zunge: Aufgeschlagen
Schild: gespalten: Feld 1: Säugetier: Kopf: gehörnt: Farbe: Silber
Schild: gespalten: Feld 1: Säugetier: Kopf: Krone: Farbe: Gold
Schild: gespalten: Feld 2: Farbe: Silber
Schild: gespalten: Feld 2: Fabelwesen: Art eines Fabelwesens: Greif
Schild: gespalten: Feld 2: Fabelwesen: Farbe: Rot
Schild: gespalten: Feld 2: Fabelwesen: Bewegung: Aufgerichtet
Schild: gespalten: Feld 2: Fabelwesen: Bewehrung: Farbe: Gold

Hierbei wird offensichtlich, dass die Begriffe in den Ketten immer vom Allgemeinen zum Detail angeordnet werden. Am Anfang der Ketten stehen die allgemeinen Aussagen wie Spaltung oder um welches Teilfeld es sich handelt. Am Ende der Ketten stehen die konkreten Ausprägungen wie Farben oder Körperhaltungen von Wappentieren.
Im Allgemeinen kann man die Struktur, in der die Begriffe angeordnet sind, als gerichteten Graphen auffassen, größtenteils sogar als Baum, wobei die adjazenten Knoten jedes Knotens diejenigen Begriffe repräsentieren, die den Startknoten der Kante genauer beschreiben. Einige Begriffe, die sehr häufig vorkommen wie zum Beispiel *Farbe*, sind der Endknoten einer Vielzahl von Kanten, hätten also mehrere Elternknoten, und verletzen damit die Baumeigenschaft, dass jeder Knoten nur eine eingehende Kante hat. Andere Begriffe, wie zum Beispiel die einzelnen Felder einer Aufteilung, können als Kindknoten Begriffe haben, die in der Baumstruktur eigentlich oberhalb kämen und erzeugen dadurch Zyklen. Ein praktisches Beispiel wäre ein geteiltes Wappen mit einem wiederum geteilten Teilfeld, wie es das Wappen der Hansestadt Rostock zeigt. In Abbildung 3.2 ist eine stark vereinfachte Version eines Begriffsgraphen mit den beschriebenen Sonderfällen zu sehen. Die einzelnen Begriffsketten sind dann Pfade im Graphen ausgehend vom definierten Startknoten oder Einstiegspunkt.
Im unter Voß' Aufsicht entwickelten Anwendungsprogramm, das mit den Begriffsketten arbeitet, wählt der Nutzer den nächsten Begriff einer Kette nur aus einer Liste von Begriffen aus, die den vorigen Begriff detaillierter beschreiben (die adjazenten Knoten im Graph). Es handelt sich also um ein „Zusammenklicken" der Wappenbeschreibung im wörtlichen Sinne. Eine eigene Eingabe von Text ist für die Nutzer weder nötig noch möglich, was orthografische Fehler verhindert. Von entscheidender Bedeutung für eine zielführende und heraldisch korrekte Arbeit mit dem System ist der Aufbau eines möglichst vollständigen Begriffsnetzes. Dieses sollte von einem oder mehreren Administratoren mit umfassendem heraldischen Wissen definiert werden, die sicherstellen müssen, dass für dasselbe Konzept immer nur ein Begriff im Graphen existiert.
Zur besseren Strukturierung und Übersichtlichkeit der Begriffswelt gibt es zusätzlich zu den in den Ketten vorkommenden *Begriffen*, die die tatsächliche Blasonierung von Wappen repräsentieren und überwiegend so auch in einer Volltextblasonierung vorkommen würden, sogenannte *Sammelbegriffe*, die mehrere gleichartige *Begriffe* zusammenfassen und gerade für Laiennutzer die Navigation im Begriffsgraphen erleichtern. Die *Begriffe* „Vogel", „Amphibien/Reptilien", „Fisch", „Säugetier" und „Fabelwesen" würden so zum Beispiel unter dem *Sammelbegriff* „Tiere,

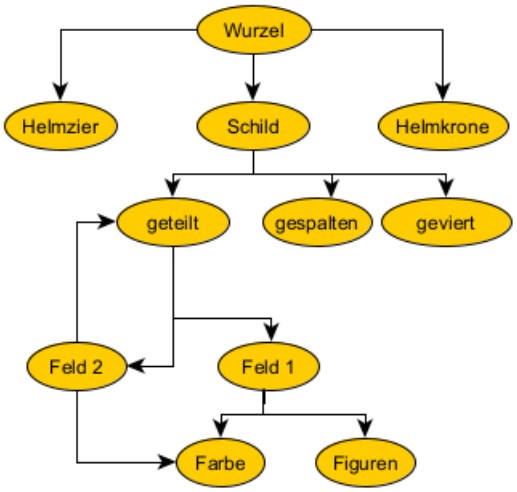


Abbildung 3.2: Ausschnitt aus einem stark vereinfachten Begriffsgraphen

auch Fabelwesen“ zusammengefasst. In der nach außen hin sichtbaren Begriffskette tauchen die Sammelbegriffe allerdings nicht auf (siehe obiges Beispiel). In der bestehenden Wappendatenbank wie auch im Folgenden steht den Sammelbegriffen der Präfix „**“ voran. Die erste Begriffskette aus dem obigen Beispiel sähe mit den Sammelbegriffen wie folgt aus (tatsächliche Begriffe fettgedruckt):

Teile des Wappens: **Schild: **mit Feldteilung: **Spaltung: **gespalten**:
Rangfolge der Felder (gespalten): **Feld 1: **ohne Feldteilung:
**Blasonierung von Schild und Feld: **Farbe und Pelzwerk:
Farbe auf Fläche (mit Figur): **Farbe: **Gold**

Zusammenfassend lässt sich feststellen, dass sich die Idee der Begriffsketten von Michael Voß und der in [HR20] beschriebene Ansatz von Hiltmann und Riechert sowohl in ihrem Modellierungsansatz als auch in den gebotenen Vorteilen zumindest ähneln. Bei beiden wird die Blasonierungssprache als Graph von eindeutig identifizierbaren Begriffen beziehungsweise Konzepten formalisiert, mithilfe dessen Wappen blasoniert werden können. Mehrsprachigkeit kann bei der Arbeit mit Begriffsketten einfach implementiert werden, indem in jedem Begriffsknoten die Bezeichnung in verschiedenen Sprachen gespeichert wird. Bei der Recherche nach Wappen können als Anfrage einzelne Begriffsketten eingegeben und damit nur nach Teilen des Wappens recherchiert werden.
Allerdings stellt sich bei den Begriffsketten dasselbe Problem, das auch schon in [HR20] genannt wird: Die Formalisierung der in Teilen ungenauen Blasonierungssprache in einem Modell für die computergestützte Arbeit ist der umgangssprachliche Knackpunkt in der digitalen Heraldik. Die vielen Sonderfälle und regionalen Unterschiede in Bezeichnungen müssen bei der Definition des Begriffsgraphen berücksichtigt werden, um einen möglichst großen Anwendungsbereich zuzulassen. Das bestehende System bietet zwar die Möglichkeit, Synonyme für Begriffe zu definieren, nach denen auch recherchiert werden kann, dennoch bleibt die Schaffung eines vollumfänglichen Systems eine schwierige Aufgabe.

Kapitel 4

Konzept

Wie schon in Kapitel 1.2 beschrieben, besteht das Hauptziel dieser Arbeit darin, die bestehende Wappendatenbank von Michael Voß zu überarbeiten, um eine einfache Nutzung für eine möglichst breite Gruppe zu ermöglichen. Dazu soll die zugrundeliegende Idee der Begriffsketten aufgegriffen und in einer Webanwendung mit modernen Technologien implementiert werden. Der erste Schritt sieht die Integration der Anwendung ins Ortschronikenportal Mecklenburg-Vorpommern[1] vor. In diesem Umfeld kann sich die Anwendung und die Arbeitsweise mit Begriffsketten in einem ausreichend großen aber dennoch überschaubaren Nutzerkreis bewähren. Nach Einarbeitung von Feedback und Behebung eventueller Probleme ist in einem weiteren Schritt der Zugang für ein breites Publikum möglich.
Das zweite Ziel dieser Arbeit ist eine Untersuchung, inwieweit Semantic-Web-Techniken für die digitale Arbeit mit Wappen eingesetzt werden können. In Kapitel 3.2 wurde bereits ein Ansatz aus der Literatur vorgestellt, bei dem Wappen mithilfe einer Ontologie beschrieben werden. Für die Graphdatenbank Neo4j, die für die neue Anwendung verwendet wird (siehe Kapitel 5.1.1), existiert ein Plugin, das den Datenimport und -export im RDF-Format ermöglicht.
Im nun folgenden Kapitel soll zunächst das Konzept für die neue Anwendung beschrieben werden. Auf dessen Implementierung wird dann im anschließenden Kapitel 5 eingegangen. Im Abschnitt 4.1 soll als erstes das von Michael Voß in [Voß21] vorgeschlagene Nutzerrollenmodell für die Arbeit mit Begriffsketten kurz vorgestellt werden. Anschließend wird in Abschnitt 4.2 auf den Funktionsumfang der bereits bestehenden Anwendung eingegangen. In Abschnitt 4.3 geht es dann um die Anforderungen und den Funktionsumfang der neu zu entwickelnden Anwendung.

4.1 Das Rollenmodell

Michael Voß schlägt in [Voß21] ein dreistufiges Rollenmodell für die Arbeit mit einer Anwendung vor, die die Idee der Begriffsketten implementiert. Dieses sieht die Rollen „Inhaltsverwalter“, „Erfasser“ und „Nutzer“ vor, deren Aufgaben im Folgenden erläutert werden:

- **Inhaltsverwalter**: Diese Rolle hat Zugriff auf alle Funktionen im System und steht in engem Kontakt zu den Entwicklern und Systemadministratoren, an die zum Beispiel gemeldete Fehler weitergeleitet werden können. Auch neu zu entwickelnde Features werden mit den Inhaltsverwaltern abgestimmt. Die Rolle ist verantwortlich für die Definition des Begriffsgraphen. Das bedeutet, vor Nutzung der Anwendung durch andere Rollen müssen zunächst die einzelnen heraldischen Begriffe mit Erklärungen und bildlichen Darstellungen eingepflegt und Verknüpfungen der Begriffe zu möglichen Ketten definiert werden.

[1] https://www.ortschroniken-mv.de/index.php/Ortschroniken_Mecklenburg-Vorpommern, zuletztabgerufenam17.08.2021

Darüber hinaus ist es Aufgabe der Inhaltsverwalter, neue Nutzer der Rolle Erfasser zuzuweisen, sofern diese dafür geeignet sind, und die Arbeit der Erfasser zu kontrollieren, damit deren Arbeitsweise über das gesamte System hinweg möglichst einheitlich ist. Die Arbeit der Inhaltsverwalter sollten nach Möglichkeit qualifizierte Heraldiker übernehmen. Gerade für die Definition des Begriffsgraphen ist das essentiell, um eine korrekte heraldische Arbeitsweise zu gewährleisten.

- **Erfasser**: Die Hauptaufgabe der Erfasser ist das Einpflegen neuer Wappen in die Datenbank mit ihren Blasonierungen als Begriffsketten sowie erläuternden Texten und Bildern. Im Kontakt mit den Inhaltsverwaltern können die Erfasser neue Begriffe vorschlagen, die ihrer Meinung nach noch im Begriffsgraph fehlen.

- **Nutzer**: Die Rolle Nutzer hat anders als die anderen Rollen nur lesenden Zugriff auf die Datenbank. Das heißt, sie kann lediglich nach Wappen recherchieren.

Für die Rollen Inhaltsverwalter und Erfasser ist zwingend eine Authentifizierung nötig, die normalen Nutzer können die Anwendung auch ohne Authentifizierung nutzen. Insgesamt eignet sich das vorgeschlagene Rollenmodell sehr gut für die Umsetzung eines Crowdsourcing-Ansatzes zur Erfassung von Wappen. In einer Anwendung könnte jeder Interessierte auf das allgemein verfügbare Wissen ohne Login, das heißt ohne Einstiegshürde, lesend zugreifen. Bei weitergehendem Interesse bestünde dann die Möglichkeit der Erstellung eines Nutzeraccounts und der selbstständigen Wappenerfassung. Die Kommunikation zwischen verschiedenen Inhaltsverwaltern und Erfassern könnte dabei über im System integrierte Kanäle (beispielsweise Messaging-Dienste) laufen.

4.2 Aktuelle Anwendung

Die aktuelle Anwendung[2] basiert auf Microsofts ASP.NET-Framework und nutzt als Datenbank Microsofts SQL Server. Weitere technische Details sollen an dieser Stelle ausgelassen werden, da diese für die Entwicklung der neuen Anwendung keine besondere Relevanz haben.
Die Anwendung greift das in Kapitel 4.1 beschriebene Rollenmodell auf. Es gibt einen Bereich mit Funktionalitäten für die Inhaltsverwalter des Systems und einen mit Funktionalitäten für die Erfasser und normalen Nutzer. Neben Optionen zur Nutzerverwaltung bilden die beiden folgenden Funktionseinheiten den Kern der Arbeit im Verwaltungsbereich:

- **Begriffseditor**: Der Begriffseditor dient zum Anlegen, Bearbeiten und Löschen von Begriffen im in Kapitel 3.3 beschriebenen Begriffsgraphen. Ebenso kann eine Kante zwischen zwei bereits bestehenden Begriffen hinzugefügt werden. Die Navigation im Begriffsgraphen erfolgt mittels Auswählen des jeweils nächsten Begriffes in der Hierarchie. Auf diese Weise wird auch im Begriffseditor die Arbeit mit Begriffsketten nachempfunden. Abbildung 4.1 zeigt die Benutzeroberfläche des Begriffseditors mit den Begriffen in der aktuellen Hierarchieebene (in diesem Fall verschiedene Farben), der gebildeten Begriffskette (**Teile des Schildes: Schild: **ohne Feldteilung: ** Blasonierung von Schild und Feld usw.) sowie Buttons zum Navigieren im Graphen („weiter“ und „zurück“), zum Bearbeiten von Begriffen und zum Hinzufügen von neuen Begriffen.

- Verwaltung von **Lokationen**: In der Anwendung werden die einzelnen Wappen einem Ort (genannt „Lokation“) zugeordnet, anhand dessen später nach den Wappen recherchiert werden kann. Für kommunale Wappen kann das zum Beispiel eine bestimmte Gemeinde sein, für Adelswappen zum Beispiel eine Gruppe von Familien. Die Lokationen werden

[2] `https://wappen.azurewebsites.net/Login.aspx`, zuletzt abgerufen am 17.08.2021

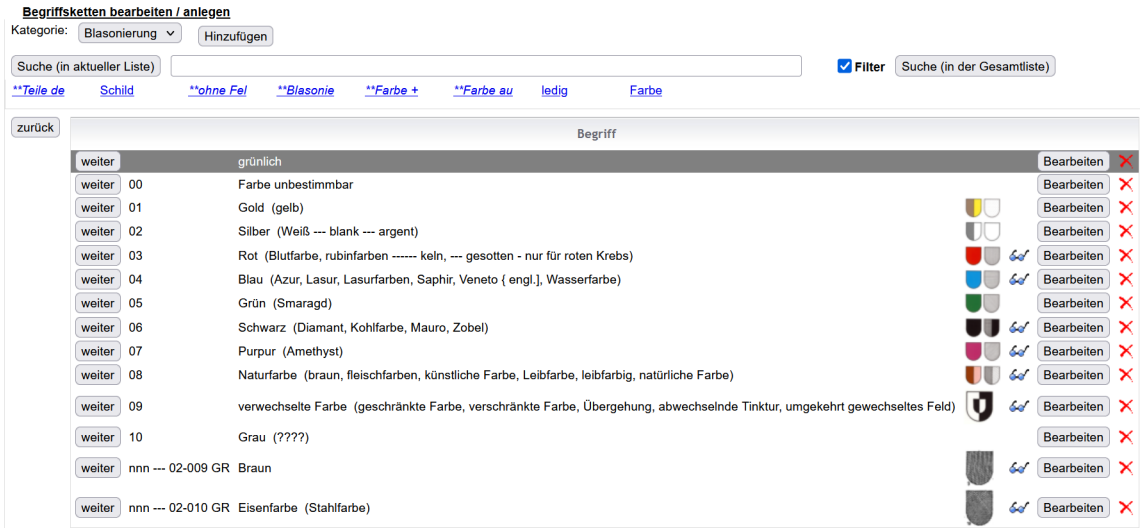


Abbildung 4.1: Screenshot der Ansicht im Begriffseditor (bestehende Anwendung)

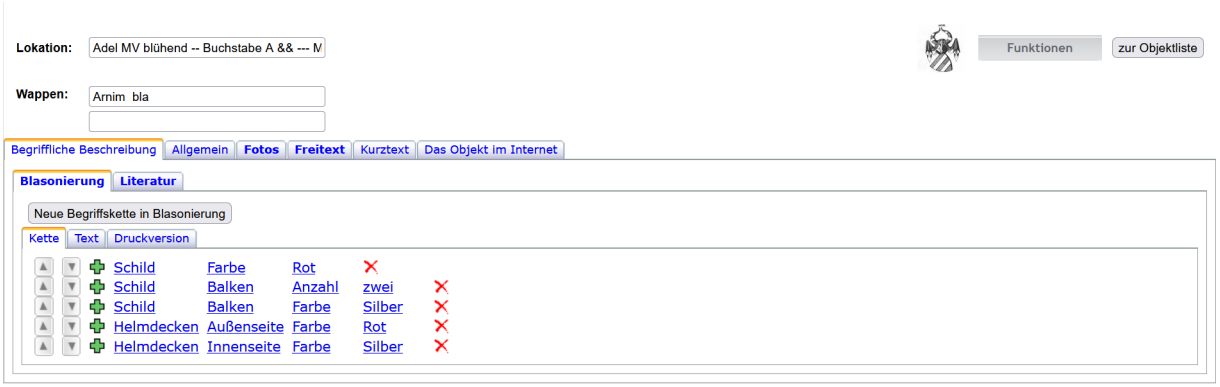


Abbildung 4.2: Screenshot der Wappenerfassung (bestehende Anwendung)

ähnlich wie die Begriffsketten in einer sechsstufigen Baumstruktur definiert. Ein Pfad von der Wurzel zum Blatt repräsentiert dann die Lokation des jeweiligen Wappens. Für die Hansestadt Rostock ist dieser Pfad beispielweise „Deutschland: Deutschland - Bundesland: Bundesland Mecklenburg-Vorpommern: Kommunale Wappen (MV): Landeswappen MV + Kreisfreie Städte: Kreisfreie Städte“.

Für die Erfasser stehen die Funktionseinheiten der Wappenerfassung und der Wappenrecherche zur Verfügung. Letztere ist auch für normale Nutzer verfügbar. Bei der Erfassung ist zunächst die Lokation des zu bearbeitenden oder neu anzulegenden Wappens auszuwählen. Anschließend können entweder die Daten der bereits erfassten Wappen bearbeitet oder ein komplett neues Wappen angelegt werden. Abbildung 4.2 zeigt die für das Wappen der Familie Arnim angelegten Begriffsketten. In den Tabs „Allgemein“, „Fotos“ und „Freitext“ können weitere Informationen über das Wappen, weitere Fotos und eine Volltextbeschreibung hinterlegt werden. Weitere Begriffsketten können über den Button „Neue Begriffskette in Blasonierung“ hinzugefügt werden. Zu Wappenrecherche können verschiedene Parameter wie Lokation, Wappenname, Begriffsketten oder einzelne Begriffe eingegeben werden. Im Anschluss werden alle Wappen als Ergebnis ausgegeben, die diese Parameter besitzen.

4.3 Neue Anwendung

Die Architektur der neu zu entwickelnden Anwendung soll einer klassischen Drei-Schichten-Architektur entsprechen, bestehend aus einer Neo4j-Graphdatenbank zur Datenhaltung, einer darauf aufsetzenden Flask-API und einem mit Angular arbeitenden Web-Frontend. Auf die verwendeten Frameworks wird in Kapitel 5.1 detaillierter eingegangen.
Die begrenzten zeitlichen und personellen Ressourcen für die vorliegende Bachelorarbeit lassen leider nicht die Entwicklung eines voll ausgereiften Systems für den sofortigen Einsatz zu. Stattdessen wird lediglich ein Prototyp vorgestellt, der die für die Arbeit mit Wappen und Begriffsketten grundlegenden Funktionalitäten (Begriffseditor, Wappenerfassung, Wappenrecherche) implementiert. Darüber hinaus gehende Features können dann ausgehend von den Ergebnissen dieser Arbeit umgesetzt werden.
In den folgenden Abschnitten werden zunächst die umzusetzenden und anschließend die nicht umgesetzten Funktionalitäten beschrieben.

4.3.1 Funktionsumfang der neuen Anwendung

Wie oben geschrieben soll sich der Funktionsumfang der neuen Anwendung auf die wesentlichen Operationen zum Anlegen, Bearbeiten, Löschen und Lesen von Wappen, Begriffsketten und Begriffen (CRUD-Operationen) beschränken. Dahingehend wird sich die Anwendung stark an der Arbeitsweise im alten System orientieren, mit aufeinanderfolgendem Anklicken des jeweils nächsten Begriffs zum Hinzufügen zu einer Begriffskette. Das User Interface wird allerdings mit neueren Technologien und Standards umgesetzt. Auch die einzelnen Funktionseinheiten entsprechen den im vorigen Kapitel 4.2 vorgestellten Begriffseditor, Wappenerfassung und Wappenrecherche. Im Folgenden wird auf die einzelnen Einheiten sowie das zugrundeliegende Datenmodell eingegangen.

Datenmodell

Im Modell werden Entitäten der Typen *CoA* (Wappen), *Chain* (Begriffskette) und *Term* (Begriff) erfasst. Wappen besitzen die Attribute *name* (Name), *location* (Lokation) und *description* (Beschreibung) sowie das Schlüsselattribut *uuid*. Begriffsketten haben lediglich eine *uuid* als Schlüssel. Die Begriffe wiederum haben die Attribute *name* (Name), *comment* (Kommentar), *hide* (Sammelbegriff) sowie eine Liste *synonyms* von Synonymen und den Schlüssel *uuid*.
Daneben beinhaltet das Modell die Beziehungstypen *HAS_CHAIN* (Hat_Kette) zwischen Wappen und Begriffsketten und *CONTAINS_TERM* (Beinhaltet_Begriff) zwischen Begriffsketten und Begriffen, beide mit dem Attribut *order* (Ordnung). Der Beziehungstyp *NEXT_TERM* (Nächster_Begriff) ist selbstreferenziell ausgehend vom Entitätstyp Begriff.
Eine Übersicht über das beschriebene Datenmodell bietet das in Abbildung 4.3 dargestellte ER-Diagramm (ohne Kardinalitäten).
Einem Wappen sind über die HAS_CHAIN-Beziehung mehrere Begriffsketten zugeordnet, einer Kette über die CONTAINS_TERM-Beziehung wiederum mehrere Begriffe. Die Ordnung gibt dabei an, an welcher Position die jeweilige Begriffskette oder der jeweilige Begriff stehen. Die Reihenfolge der Begriffsketten hat also eine Bedeutung. Beispielweise sollten alle Begriffsketten, die ein bestimmtes Wappentier beschreiben, der Übersicht halber aufeinander folgen. Ebenso gibt die Position des Begriffs an, an welcher Stelle in der Begriffskette der Begriff steht.
Über die NEXT_TERM-Beziehung werden einzelnen Begriffe miteinander zum verwendeten Begriffsgraphen verknüpft. Das hide-Attribut ist ein Boolescher Wert und gibt an, ob es sich um eine Sammelbegriff (siehe Kapitel 3.3) handelt.

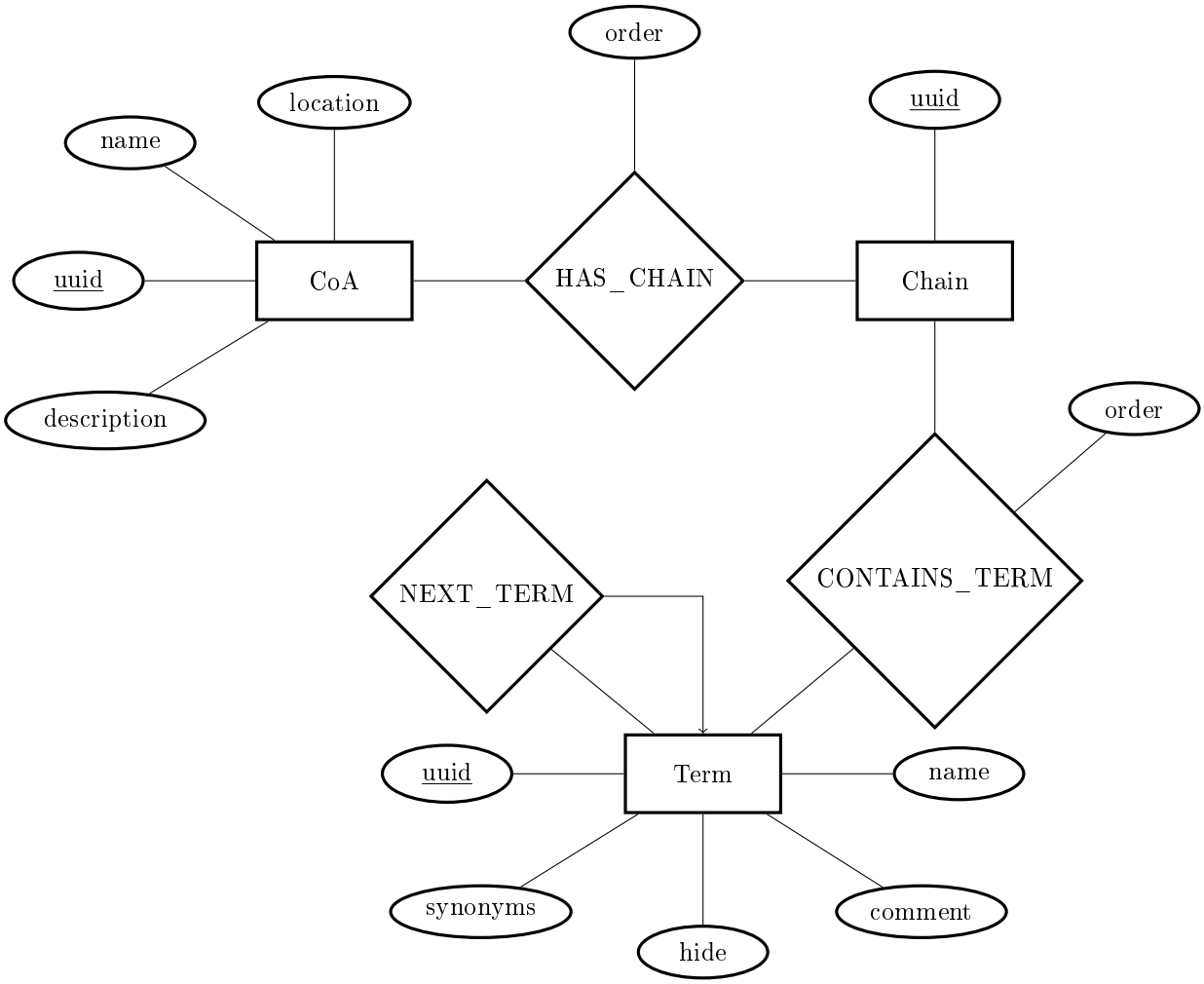


Abbildung 4.3: ER-Diagramm des Datenmodells für die neue Anwendung

Begriffseditor

Der Begriffseditor dient zur Manipulation des Begriffsgraphen, das heißt die Neuanlage, Bearbeitung und das Löschen von Begriffen und deren Verknüpfungen untereinander. Die Navigation im Graphen, also das Erreichen der Stelle, die manipuliert werden soll, geschieht durch aufeinanderfolgendes Auswählen des jeweils nächsten Begriffes auf dem Weg zur zu bearbeitenden Stelle. Beginnend mit einem der Einstiegspunkte des Graphen (Knoten ohne Vorgänger) kann der Nutzer in jedem Schritt einen der Nachfolger oder Vorgänger des aktuellen Knotens auswählen, um von diesem aus dann weiter zu navigieren. In diesem Punkt unterscheidet sich die Funktionsweise vom gegenwärtig genutzten System. Dort ist immer nur die Auswahl aus den Nachfolgern möglich. Beim Bearbeiten des Graphen wurde bisher also immer implizit eine Begriffskette verfolgt, was nun nicht mehr zwangsläufig der Fall ist. Neben diesem Auswählen des nächsten Begriffes ist auch ein direkter Sprung zu einem der zuvor ausgewählten Begriffe möglich, um effizientes Arbeiten zu gewährleisten.
An einem bestimmten Punkt im Graphen angekommen, stehen dem Nutzer verschiedene Optionen der Bearbeitung des Graphen zur Verfügung:

- Neuanlage eines Begriffes als Vorgänger oder Nachfolger des aktuellen Begriffes mit den in 4.3.1 genannten Attributen
- Bearbeiten der Daten des aktuellen Begriffes
- Löschen des aktuellen Begriffes, sofern dieser ein Blattknoten im Graphen und nicht Teil einer Begriffskette ist.
- Verknüpfung mit einem bereits bestehenden Begriff als Vorgänger oder Nachfolger beziehungsweise Entfernen der Verknüpfung mit einem Vorgänger oder Nachfolger

Wappenerfassung

Im Rahmen der Wappenerfassung werden neue Wappen mit ihren Daten (Name, Ort, Beschreibung) in die Datensammlung aufgenommen. Begriffsketten können, müssen aber nicht, für die Wappenbeschreibung angelegt werden. Ebenso können die erfassten Daten für schon existierende Wappen bearbeitet werden. Dies umfasst sowohl die Metadaten als auch die Begriffsketten, die gelöscht, neu angelegt oder deren Postion verschoben werden kann.
Für einen schnellen Überblick über die bereits existierenden Wappen empfiehlt sich eine Übersicht über alle erfassten Wappen mit Filter- und Sortieroptionen.

Wappenrecherche

Bei der Recherche nach Wappen stehen dem Nutzer verschiedene Suchparameter zur Verfügung. Neben den Metadaten Name und Lokation des Wappens kann der Nutzer eine Reihe von Begriffsketten bilden, die Teil der Wappenbeschreibung sein sollen. Diese müssen nicht zwangsläufig vollständig sein, das heißt bei einem Einstiegspunkt des Graphen beginnen, sondern können auch nur eine Teilkette sein. Zusätzlich zu den Begriffsketten kann auch eine Liste von Stichwörtern angegeben werden, die in irgendeiner der Begriffsketten vorkommen sollen. Der Nutzer könnte also auch nach dem Inhalt von Wappen suchen, ohne Begriffsketten angeben zu müssen.
Das Rechercheergebnis wird dem Nutzer anschließend in übersichtlicher Form präsentiert und dieser kann einzelne Wappen aus dem Ergebnis auswählen, um sich über deren Details zu informieren.

4.3.2 Nicht umgesetzte Funktionalitäten

Neben den im vorigen Abschnitt vorgestellten Kernfunktionalitäten gibt es eine Reihe von Features, die auf dem Weg zu einem nutzungsbereiten System umgesetzt werden müssen, beziehungsweise sollten, die aber wegen der begrenzten Ressourcen im Rahmen dieser Bachelorarbeit nicht umgesetzt werden können. Im Folgenden sollen diese Punkte kurz erläutert werden.

Rollenmodell

Das in Kapitel 4.1 vorgestellte Rollenmodell ist unerlässlich für eine nutzbare Anwendung. Anderenfalls könnte jeder beispielsweise den Begriffsgraphen manipulieren und damit das System unnutzbar machen. Zur Authentifizierung ist der Login mit einem registrierten Nutzerkonto nötig.

Einbindung von Bildern

Ohne die Möglichkeit, zu den erfassten Wappen auch Bilder hochladen und später ansehen zu können, fehlt der Anwendung ein essentieller Bestandteil. Dem geübten Heraldiker würde die reine Blasonierung mit Begriffsketten vielleicht noch ausreichen, um sich das Wappen vorstellen zu können. Für Laien, die den Großteil der Nutzergruppe darstellen, ist eine bildliche Darstellung allerdings unerlässlich, um das Aussehen der Suchergebnisse mit dem des ihnen vorliegenden Wappens vergleichen zu können.
Zusätzlich zu Bildern der Wappen sollten auch für die einzelnen Begriffe beschreibende Darstellungen verfügbar gemacht werden, um die Bedeutung der Begriffe gerade für Laien besser verständlich zu machen.
Anders als bekannte relationale Datenbankmanagementsysteme bietet die verwendete Neo4j-Plattform keine Möglichkeit, Bilder beziehungsweise allgemein Multimediadaten zu speichern. Stattdessen müsste zur Umsetzung dieses Features eine URL oder ein Dateipfad in der Datenbank gespeichert werden, die dann auf eine externe Ressource oder eine Datei im Dateisystem des Servers verweisen.

Lokationen

In der aktuell genutzten Anwendung werden die Lokationen (siehe Kapitel 4.2) ähnlich hierarchisch verwaltet wie die Begriffe. In der neuen Anwendung wird die Lokation zunächst nur als Attribut des Wappens gespeichert und ist frei wählbar, ohne dass die Inhaltsverwalter die verwendeten Lokationen verwalten könnten.

Optimierung für mobile Geräte

Die Verwendung von Responsive Design zur Gestaltung der Webanwendung ist empfehlenswert, da sie auch mit mobilen Endgeräten genutzt wird. Gerade Ortsforscher, die zum Beispiel in einer Kirche auf alte Dokumente mit heraldischem Inhalt stoßen, haben nicht immer Zugang zu einem PC oder Laptop sondern möchten von ihrem Smartphone aus auf die Wappensammlung zugreifen. Für diesen Fall sollte einerseits das Frontend auch für mobile Geräte optimiert sein, andererseits die Kommunikation mit dem Server möglichst Daten sparend sein.
Angular bietet an dieser Stelle das `layout`-Paket[3] an, mit dem vergleichsweise einfach eine responsive Benutzeroberfläche geschaffen werden kann.

[3] `https://material.angular.io/cdk/layout/overview`, zuletzt abgerufen am 28.08.2021

Migration der Daten aus dem bestehenden System

Vor Beginn der Nutzung der neuen Anwendung sollten zunächst die Daten aus dem Altsystem ins neue System migriert werden. Anderenfalls müssten die etwa 4300 Begriffe und 2000 Wappen manuell neu erfasst werden. Dazu muss zunächst das bestehende relationale Datenmodell evaluiert und die tatsächlich zu übernehmenden Daten identifiziert werden. Diese Daten müssen anschließend in die Neo4j-Datenbank übernommen werden. Der Import von CSV-Dateien ist in Neo4j vergleichsweise einfach umzusetzen[4]. Allerdings sind auch andere Vorgehensweisen denkbar.

Export der Daten zur Nutzung im Semantic Web

In [HR20] stellen die Autoren ihre Idee einer heraldischen Ontologie vor (siehe Kapitel 3.2). Ein konkretes Ergebnis, eine umfassende Ontologie, die auch breite Verwendung finden könnte, ist allerdings bis zum heutigen Zeitpunkt nicht veröffentlicht worden. Nach einem ähnlichen Ansatz stellt die Strukturierung heraldischer Begriffe nach Voß in einem Begriffsgraphen die Zusammenhänge zwischen den heraldischen Konzepten (Begriffe) dar. Daraus ließe sich mit der richtigen Konfiguration direkt eine Ontologie erzeugen, die auch genutzt werden kann. Für Neo4j existiert genau für solche Anwendungsfälle das *Neosemantics*-Plug-in[5], mit dem Daten in RDF-Format im- wie auch exportiert werden können.

[4] `https://neo4j.com/docs/cypher-manual/current/clauses/load-csv/`, zuletzt abgerufen am 28.08.2021

[5] `https://neo4j.com/labs/neosemantics/4.0/`, zuletzt abgerufen am 28.08.2021

Kapitel 5

Implementierung

Wie in Kapitel 4.3 schon kurz angedeutet, entspricht der Prototyp der neuen Anwendung einer Drei-Schichten-Architektur. Zur Speicherung von Begriffen, Wappen und Begriffsketten wird eine Neo4j-Graphdatenbank verwendet. Der Nutzer interagiert mit einem Web-Frontend, das mit dem Angular-Framework implementiert wurde. Zur Kommunikation zwischen beiden Komponenten wird eine REST-API verwendet, die im Python-Webframework Flask geschrieben wurde.
Im folgenden Kapitel werden zunächst die verwendeten Technologien und Frameworks, Neo4j, Flask und Angular, genauer vorgestellt. Im Anschluss wird die Umsetzung des Prototyps in den einzelnen Architekturkomponenten erläutert.

5.1 Verwendete Technologien

Die nachfolgenden Teilkapitel geben einen kurzen Überblick über die Funktionsweise der Graphdatenbank Neo4j mit seiner Anfragesprache Cypher, des Python-Webframeworks Flask und Googles Webframework Angular.

5.1.1 Neo4j

Neo4j[1] ist eine in Java und Scala implementierte Graphdatenbank, die mit dem in Kapitel 2.3 vorgestellten Property-Graph-Modell zur Speicherung von Daten arbeitet. Als Entwicklungswerkzeuge stehen *Neo4j Desktop* und *Neo4jBrowser* zur Verfügung. Neo4j-Desktop ist ein Management-Tool, mit dem Entwickler mehrere Datenbankinstanzen, sowohl lokal als auch remote, einfach über eine Benutzeroberfläche verwalten können. Auch das Installieren von diversen Plug-ins ist darüber möglich. Bei Neo4j-Browser handelt es sich um ein Werkzeug zur Interaktion mit dem Inhalt einer Neo4j-Datenbank. Nutzer können Anfragen in Cypher (siehe unten) formulieren und sich deren Ergebnisse als visualisierten Graphen anzeigen lassen. Abbildung 5.1 zeigt beispielsweise das Ergebnis einer Anfrage im Neo4j-Browser, die alle Knoten mit dem Label „Term“ zurückgibt.
Zum Zugriff auf Neo4j-Datenbanken stellen die Entwickler offizielle Bibliotheken für verschiedene Programmiersprachen (unter anderem Java, JavaScript, Python) zur Verfügung. Für eine Reihe weiterer Sprachen existieren Open-Source-Lösungen. Mithilfe dieser Bibliotheken ist eine einfache Einbindung von Neo4j-Datenbanken in eine Anwendung möglich.
Für Anfragen verwendet Neo4j *Cypher*, eine eigens entwickelte deklarative Anfragesprache, die

[1] `https://neo4j.com/`, zuletzt abgerufen am 19.08.2021

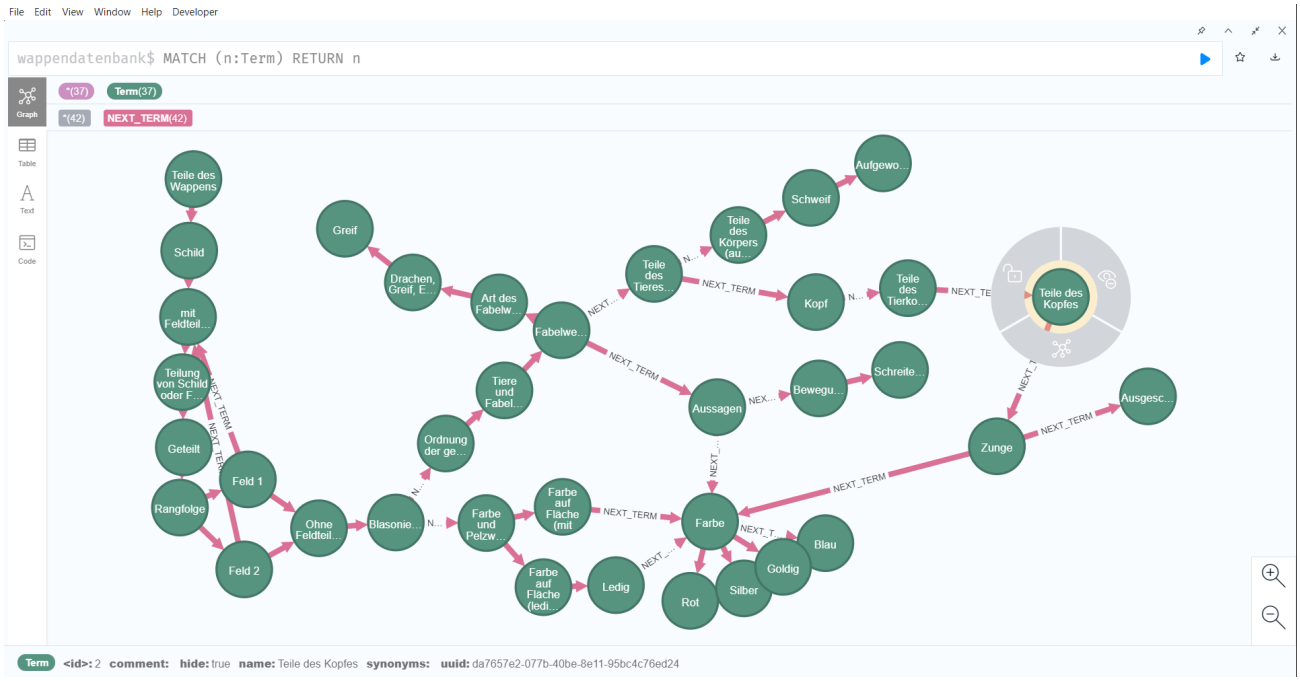


Abbildung 5.1: Screenshot des Neo4j-Browsers mit dem Begriffsgraphen

```
1  ()          // anonymer Knoten ohne Label oder Variable
2  (t:Term)    // Knoten mit der Variable t und dem Label Term
3  (:Term)     // Knoten ohne Variable mit dem Label Term
```

Abbildung 5.2: Cypher: Möglichkeiten zur Definition von Knoten

in einigen Sprachkonstrukten SQL ähnelt. Im Folgenden wird auf einige grundlegende Sprachkonzepte und Operationen von Cypher näher eingegangen.

Nodes

Knoten, englisch Nodes, werden in Cypher in Klammern eingefasst. Für Knoten können Variablen vergeben werden, um später in der Anfrage noch darauf zugreifen zu können. Ebenso können Knoten aber auch anonym, das heißt ohne Variable angegeben werden. Mithilfe von Knotenlabels nach einem Doppelpunkt kann die Suche auf bestimmte Knoten begrenzt werden. Abbildung 5.2 zeigt verschiedenen Möglichkeiten zur Angabe von Knoten in Cypher.

Relationships

Die Beziehungen, englisch Relationships, zwischen den Knoten werden in Cypher über Pfeile (--> oder <--) für gerichtete und Linien (--) für ungerichtete Beziehungen dargestellt. Variablen und Labels für Beziehungen werden ähnlich wie bei Knoten in eckigen Klammern zwischen den --Zeichen geschrieben. Beispiele sind in Abbildung 5.3 zu sehen.

```
1    // gerichtete Beziehung ohne Label oder Variable
2    -->
3    // gerichtete Beziehung mit der Variable nt und dem Label
     NEXT_TERM
4    -[nt:NEXT_TERM]->
5    // ungerichtete Beziehung mit dem Label NEXT_TERM
6    -[:NEXT_TERM]-
```

Abbildung 5.3: Cypher: Möglichkeiten zur Definition von Relationships

```
1    // Knoten mit dem Attribut name
2    (t:Term {name: 'Schild'})
3    // Beziehung mit dem Attribut order
4    -[ct:CONTAINS_TERM {order: 1}]->
```

Abbildung 5.4: Cypher: Angabe von Properties für Nodes und Relationships

Properties

Node- und Relationship-Properties können in Cypher als Name-Wert-Paare in geschweiften Klammern innerhalb der Klammern von Nodes oder Relationships angegeben werden. In Abbildung 5.4 sind beide Fälle dargestellt.

Anfragen nach bestimmten Strukturen im Graph

Die `MATCH`-Klausel ermöglicht die Suche nach bestimmten Strukturen im Graph, einzelne Nodes und Relationships aber auch komplexe Pfadstrukturen. Mithilfe der `WHERE`-Klausel kann das von `MATCH` gefundene Ergebnis eingeschränkt werden, beispielsweise auf Knoten, die einen bestimmten Attributwert oder eine bestimmte Relationship haben. In der `RETURN`-Klausel wird schließlich angegeben, welche Nodes, Relationships oder Properties die Anfrage zurückliefern soll. Dabei kann auf die zuvor definierten Variablen zugegriffen werden. In Abbildung 5.5 ist eine Anfrage zu sehen, die nach allen Knoten mit den Labels `CoA` und `Chain` sucht, die über eine Relationship des Typs `HAS_CHAIN` miteinander verbunden sind. Dabei darf der Name des `CoA`-Knotens nicht mit `'Hanse'` beginnen und der `Chain`-Knoten muss mindestens eine `CONTAINS_TERM`-Relationship zu einem Knoten mit dem Label `Term` haben. Als Ergebnis werden die so gefundenen `CoA`-Knoten zurückgegeben.

Anlegen, Update und Löschen von Nodes und Relationships

Über die `CREATE`-Klausel können neue Nodes und Relationships mit Labels und Properties angelegt werden. Diese gesetzten Werte können später über die `SET`-Klausel verändert und über die `REMOVE`-Klausel wieder entfernt werden. Mittels `DELETE` können Nodes und Relationships wieder gelöscht werden. Abbildung 5.6 zeigt Beispiele für die genannten Fälle.

```
1    MATCH (coa:CoA)-[:HAS_CHAIN]->(c:Chain)
2    WHERE NOT coa.name STARTS WITH 'Hanse'
3    AND EXISTS((c)-[:CONTAINS_TERM]->(:Term))
4    RETURN coa
```

Abbildung 5.5: Cypher: Beispiel für eine Anfrage

```
1  // Neuanlegen zwei neuer Nodes mit einer Relationship zwischen
   ihnen
2  CREATE (parent:Term {name: 'Schild'})-[:NEXT_TERM]->(child:Term
   {name: 'Farbe'})
3  // Matchen eines Nodes mit der entsprechenden Eigenschaft
4  MATCH (t:Term)
5  WHERE name = 'Schild'
6  // Verändern des name-Properties
7  SET t.name = 'Wappenschild'
8  // Entfernen des name Properties
9  REMOVE t.name
10 // Löschen des zuvor gematchten Nodes mitsamt der verbundenen
   Relationships
11 DETACH DELETE t
```

Abbildung 5.6: Cypher: Beispiele für `CREATE`-, `SET`-, `REMOVE`- und `DELETE`-Klauseln

5.1.2 Flask

Flask[2] ist ein in Python geschriebenes Micro-Webframework. Das „Micro“ bezeichnet hierbei nach eigener Aussage ([Fla]), dass Flask versuche, den „Kern einfach aber erweiterbar“ zu halten. Flask zwinge dem Entwickler keine bestimmte Vorgehensweise auf, indem es viele Funktionalitäten, wie zum Beispiel Datenbankanbindung oder Validierung, nicht selbst implementiere. Stattdessen stelle es nur die nötigsten Funktionalitäten bereit. Für alles darüber hinaus gäbe es andere Bibliotheken, die diese Aufgaben übernehmen könnten. Diese könnten mittels Erweiterungen einfach in die Anwendung eingebunden werden.
Aufgrund der geringen Komplexität eignet sich Flask gut, um schnell für einfache Anwendungsfälle Webanwendungen zu gestalten. Im Beispiel der Wappendatenbank konnte mithilfe von Flask etwa schnell eine REST-API entwickelt werden, die die Schnittstelle zwischen Frontend und Datenbank bildet. Abbildung 5.7 zeigt ein Minimalbeispiel einer Flask-Anwendung, die einen HTTP-Endpunkt anbietet. Erklärungen dazu finden sich in den Kommentaren des Quelltextes.

5.1.3 Angular

Bei Angular[3] handelt es sich um ein von Google entwickeltes TypeScript[4]-Framework zur Entwicklung von Single-Page-Webanwendungen. Grundbaustein von Angular-Anwendungen sind die sogenannten *Components* (deutsch Komponenten, zur Erklärung siehe unten), elementare UI-Elemente, die zusammen die Benutzeroberfläche bilden. Darüber hinaus beinhaltet das Framework eine Reihe von Bibliotheken, die die Umsetzung gängiger Aufgaben in der Webentwicklung erleichtern, wie beispielsweise Routing, Formular-Validierung, Kommunikation über HTTP oder Animationen. Im Folgenden sollen einige der wesentlichen Konzepte von Angular kurz erklärt werden (Vgl. [Goo]).

Components

Components sind die wesentlichen Bausteine, aus denen eine Angular-Anwendung besteht. Sie bestehen aus einer TypeScript-Klasse mit dem `@Component()`-Decorator, einer HTML-

[2] `https://flask.palletsprojects.com/en/2.0x`, zuletzt abgerufen am 24.08.2021
[3] `https://angular.io`, zuletzt abgerufen am 25.08.2021
[4] `https://www.typescriptlang.org`, zuletzt abgerufen am 25.08.2021

```python
# Import der nötigen Klassen/Funktionen
from flask import Flask, jsonify

# Erzeugen einer Instanz der Flask-Klasse
app = Flask(__name__)

# Definition des Endpunktes mit URL, die die Funktion aufruft und angebotenen HTTP-Methoden
@app.route("/path/to/endpoint", methods=["GET"])
def myEndpoint():
  # Lesen des Parameters "param" aus dem Query-String des Requests
  param = request.args["param"]

  # hier kann zum Beispiel ein Datenbankzugriff geschehen, im Beispiel wird nur ein Dictionary definiert
  result = {"Parameter": "param", "Text": "Hello World!"}

  # Als Response wird das zuvor definierte Dictionary in JSON übersetzt
  return jsonify(result)
```

Abbildung 5.7: Flask: Minimalbeispiel für eine REST-API mit einem Endpunkt

```typescript
@Component({
  selector: 'app-hello-world',
  templateUrl: './hello-world.component.html',
  styleUrls: ['./hello-world.component.css']
})
export class HelloWorldComponent {
  // hier steht der Code für die Logik der Komponente
}
```

Abbildung 5.8: Angular: Minimalbeispiel einer Component-Klasse

Datei, die das User Interface der Komponente beschreibt, und einer CSS-Datei zum Styling der Komponente. In Abbildung 5.8 ist ein Minimalbeispiel für die Typescript-Klasse einer Komponente zu sehen. Im `@Component()`-Decorator finden sich einerseits Verweise auf die zugehörigen HTML- und CSS-Dateien (`templateUrl` und `styleUrls`) und andererseits der CSS-Selektor der Komponente, mit dem diese aus anderem HTML-Code heraus instanziiert werden kann. Die Komponente aus Abbildung 5.8 würde so zum Beispiel mit `<app-hello-world></app-hello-world>` genutzt werden.

HTML-Templates

Wie zuvor beschrieben bestehen Komponenten unter anderem aus einem HTML-Template, das das User Interface der Komponente auszeichnet. Angular erweitert die Standard-HTML-Syntax um einige Aspekte, die dynamische Werte im UI erlaubt. Bei Änderung der Werte wird auch automatisch das UI neu gerendert. Einige Beispiele für dieses Vorgehen sind in Abbildung 5.9 zu finden. Die Erklärung steht dabei in den Kommentaren.

```typescript
// app.component.ts

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  // wird in HTML als dynamischer Text gerendert
  title = 'HelloWorld';
  // wird als Attributwert an HTML-Element übergeben
  fontColor = 'blue';
  // abhängig vom Boolschen Wert wird HTML-Element angezeigt
  showText = false;
  // für jedes Listenelement wird ein HTML-Element angezeigt
  listItems = ['Wert 1', 'Wert 2', 'Wert 3'];

  // Funktion wird beim Button-Klick aufgerufen
  toggleBoolean() {
    this.showText = !this.showText;
  }
}
```

```html
<!-- app.component.html -->

<!-- Attributwert color in eckigen Klammern hängt an der
Variable aus .ts-Datei -->
<!-- Inhalt der geschweiften Klammern wird aus .ts-Datei
interpoliert -->
<p [style.color]="fontColor">Diese Anwendung heißt:
{{title}}</p>

<!-- Mit *ngFor wird für jedes Listenelement eines der
HTML-Elemente gerendert -->
<p *ngFor="let item of listItems">{{item}}</p>

<!-- Event in Klammern gibt die aufgerufene Funktion an -->
<button (click)="toggleBoolean()">Klick mich</button>

<!-- Mit *ngIf kann die Anzeige des Elements an eine Bedingung
geknüpft werden -->
<p *ngIf="showText">Anzeige abhängig von boolschem Wert</p>
```

Abbildung 5.9: Angular: Beispiele für modifizierte HTML-Syntax

Material Design

Material Design[5] hat nicht direkt etwas mit Angular zu tun. Eher handelt es sich dabei um eine von Google entwickelte Designsprache für User Interfaces, die den Fokus auf intuitive Benutzbarkeit legt. Dies wird über die Verwendung verschiedener Komponenten mit fest definiertem Verhalten und Zuständen erreicht. Eine Übersicht über diese Komponenten ist auf `https://material.io/components` (zuletzt abgerufen am 25.08.2021) zu finden. Google bietet für die Plattformen Android, iOS, Flutter und das Web Bibliotheken an, die die Verwendung von Material-Design-Komponenten in der eigenen Anwendung möglich machen. Für Angular existiert eine Übersicht über zur Verfügung stehender Komponenten unter `https://material.angular.io/components/categories` (zuletzt abgerufen am 25.08.2021).

5.2 Entwicklung der neuen Anwendung

In den folgenden Unterkapiteln wird die Umsetzung der in Kapitel 4.3 vorgestellten Funktionseinheiten beschrieben. In Kapitel 5.2.1 geht es zunächst um die Umsetzung des Datenmodells in Neo4j, in Kapitel 5.2.2 um die Datenbankanfragen, die in den HTTP-Endpunkten der Flask-Schnittstelle gestellt werden, und in Kapitel 5.2.3 um die Umsetzung der Funktionseinheiten aus Kapitel 4.3 mit Angular-Material-Design-Elementen.

5.2.1 Neo4j-Datenbank

Das verwendete Neo4j-Datenmodell entspricht im Wesentlichen dem ER-Modell aus Kapitel 4.3.1. Die Entitäts- und Beziehungstypen aus dem ER-Modell werden in Neo4j zu Node- und Relationship-Labels. Die Attribute können als Properties der entsprechenden Nodes und Relationships übernommen werden. Constraints und Indexe werden im beschriebenen Prototyp nicht verwendet, sollten aber im Schritt zum Produktivbetrieb noch Anwendung finden, beispielsweise um die Existenz und Eindeutigkeit der UUID-Properties zu gewährleisten oder um das Suchverhalten zu optimieren. Abbildung 5.10 veranschaulicht das Datenmodell in Neo4j mit einem Screenshot aus dem Neo4j-Browser. Die Knoten in Rot stellen die einzelnen Wappen dar, die jeweils mehrere Begriffsketten (in Gelb) besitzen. An der Kante ist jeweils die Order-Property der Relationship zu finden. Die Begriffsketten wiederum sind mit allen zugehörigen Begriffen (in Grün) verbunden.

5.2.2 Flask-API

Die Schnittstelle zwischen Frontend und Datenbank beinhaltet gegenwärtig elf HTTP-Endpunkte, die mit Cypher-Anfragen lesend oder schreibend auf die Datenbank zugreifen. Zum Zugriff wird der offizielle Neo4j-Python-Treiber[6] genutzt. Dieser kann mit `pip install neo4j` installiert und mit `from neo4j import GraphDatabase` in ein Python-Programm eingebunden werden.
Das Schema der HTTP-Endpunkte ist immer gleich: Zunächst werden die Request-Parameter oder der Request-Body in Variablen eingelesen. Anschließend wird eine Datenbank-Anfrage gestellt. Das Ergebnis der Anfrage wird dann serialisiert und als JSON im Response-Body zurückgeschickt. Zur Serialisierung existieren die drei Funktionen aus Abbildung 5.11, die Node-Objekte des Anfrageergebnisses in Python-Dictionaries umwandeln. Die Listen in den Dictionaries beinhalten jeweils die Begriffsketten zu einem Wappen, die Begriffe zu einer Begriffskette

[5] `https://material.io/`, zuletzt abgerufen am 25.08.2021

[6] `https://neo4j.com/docs/python-manual/current/` und `https://neo4j.com/docs/api/python-driver/current/`, beides zuletzt abgerufen am 26.08.2021

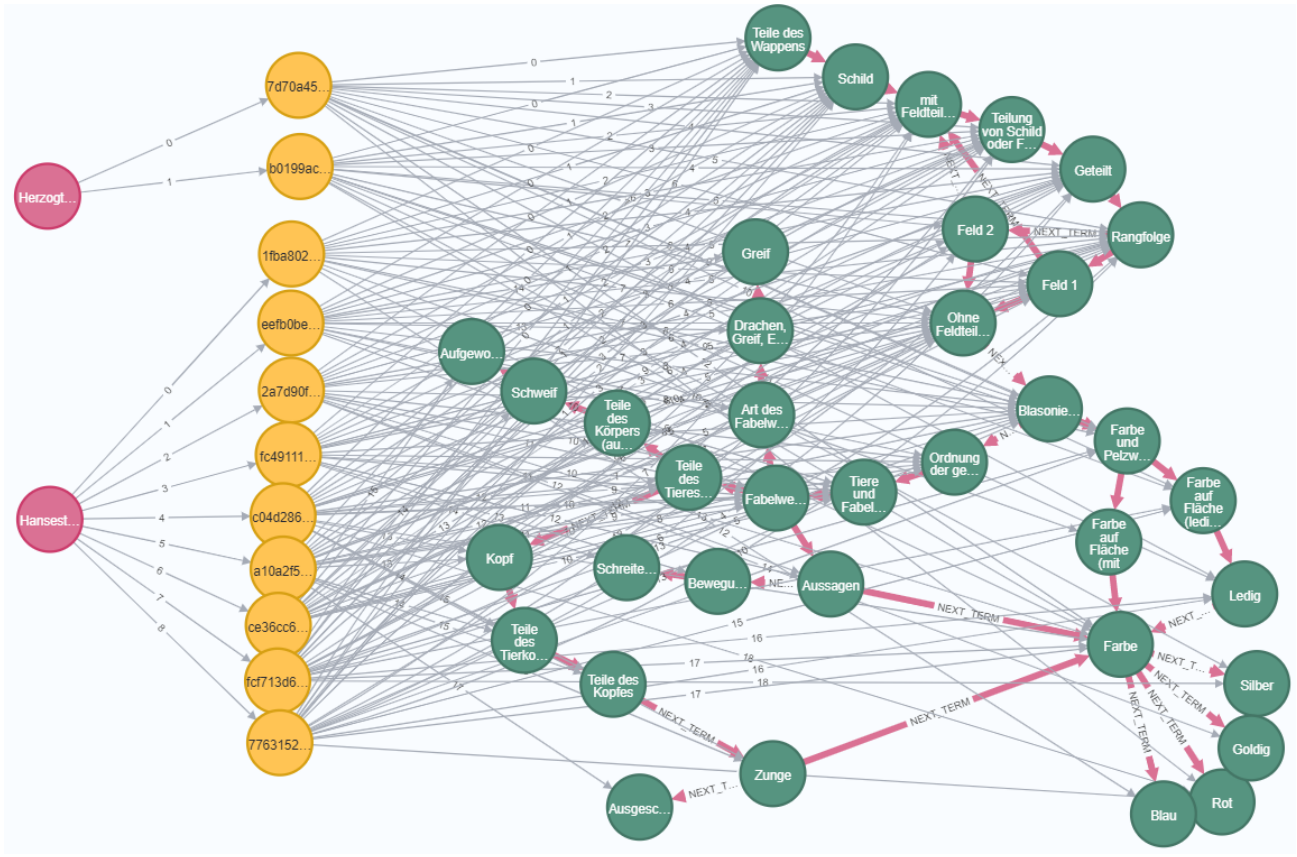


Abbildung 5.10: Veranschaulichung der Datenbank im Neo4j-Browser

```
# Funktion zur Serialisierung von Wappen
def serialize_coa (coa):
  return {
    'uuid': coa['uuid'],
    'name': coa['name'],
    'description': coa['description'],
    'location': coa['location'],
    'containedChains': []
  }
# Funktion zur Serialisierung von Begriffsketten
def serialize_chain (chain):
  return {
    'uuid': chain['uuid'],
    'containedTerms': []
  }
# Funktion zur Serialisierung von Begriffen
def serialize_term (term):
return {
  'uuid': term['uuid'],
  'name': term['name'],
  'synonyms': term['synonyms'],
  'hide': term['hide'],
  'comment': term['comment'],
  'children': [],
  'parents': []
}
```

Abbildung 5.11: Flask-API: Funktionen zur Serialisierung von Wappen, Begriffsketten und Begriffen

sowie die Vorgänger- und Nachfolgerbegriffe eines Begriffes. In den Funktionen selbst werden sie zunächst leer gelassen und erst später mit den entsprechenden Objekten befüllt.
Im Folgenden sollen nun die einzelnen Endpunkte mit erwarteten Eingabeparametern/-daten und ihrem Zweck kurz beschrieben werden. Zusätzlich ist jeweils die Cypher-Anfrage angegeben.

GET ./termeditor/allTerms

Dieser Endpunkt ist vergleichsweise simpel. In der Anfrage werden lediglich alle Nodes mit dem Label `Term` nach dem Namen des Begriffs sortiert gefunden und dann als Liste im Response-Body zurückgegeben.

Listing 5.1: Anfrage zu `/termeditor/allTerms`

```
MATCH (term:Term)
RETURN term
ORDER BY term.name
```

GET ./termeditor/firstTerms

Die Anfrage liefert alle Einstiegspunkte in den Begriffsgraphen mit ihren unmittelbaren Nachfolgern, also alle Begriffe, die keinen Vorgängerbegriff haben.

Listing 5.2: Anfrage zu `/termeditor/firstTerms`

```
MATCH (start:Term)
WHERE NOT (:Term)-[:NEXT_TERM]->(start)
OPTIONAL MATCH (start)-[:NEXT_TERM]->(child:Term)
RETURN start, collect(child) AS children
ORDER BY start.name
```

GET ./termeditor/updateListsOfClicked

Diese Funktion bekommt im Query-String zwei Parameter übergeben: eine Begriffs-UUID und einen Boolschen Wert, der bestimmt, welche von zwei Anfragen an die Datenbank gestellt wird. Abhängig von diesem Wert liefert die Anfrage entweder die Vorgänger oder Nachfolger des Begriffes mit der übergebenen UUID zurück.

Listing 5.3: Anfragen zu `/termeditor/updateListsOfClicked`

```
// $uuid ist die im Query-String übergebene UUID

// Abfrage für Nachfolger
MATCH (clicked:Term)
WHERE clicked.uuid = $uuid
OPTIONAL MATCH (clicked)-[:NEXT_TERM]->(childOfClicked:Term)
RETURN collect(childOfClicked) AS newItems

// Abfrage für Vorgänger
MATCH (clicked:Term)
WHERE clicked.uuid = $uuid
OPTIONAL MATCH (parentOfClicked:Term)-[:NEXT_TERM]->(clicked)
RETURN collect(parentOfClicked) AS newItems
```

GET/POST ./coaeditor/allCoA

Während alle anderen Endpunkte nur `GET` oder `POST` unterstützen, kann dieser mit beiden Methoden angesprochen werden. Der `POST`-Request erwartet als Daten im Body eine Liste mit Wappen-UUIDs. Die Abfrage liefert in beiden Fällen eine Liste von Wappen mit ihren Begriffsketten und deren beinhalteten Begriffen. Dabei sind sowohl die Begriffsketten der einzelnen Wappen als auch die Begriffe der einzelnen Begriffsketten nach dem `order`-Attribut der jeweils verbindenden Relationship geordnet. Während der `GET`-Request alle in der Datenbank befindlichen Wappen zurückgibt, beschränkt der `POST`-Request diese auf die Wappen mit UUIDs in der übergebenen Liste. Die Anfragen für beide Methoden unterscheiden sich dahingehend in einer Zeile.

Listing 5.4: Anfrage zu `/coaeditor/allCoA`

```
MATCH (coa:CoA)-[hc:HAS_CHAIN]->(chain:Chain)

// diese Zeile steht nur in der Anfrage des POST-Requests
// $coaUUIDs ist die Liste mit Wappen-UUIDs aus dem Request-Body
WHERE coa.uuid IN $coaUUIDs

WITH coa, chain ORDER BY hc.order
WITH coa, collect(chain) AS chains
UNWIND chains AS chain
OPTIONAL MATCH (chain)-[ct:CONTAINS_TERM]->(term:Term)
```

```
11  WITH coa, chain, term ORDER BY ct.order
12  WITH coa, chain{.*, terms: collect(term)}
13  return coa, collect(chain) AS chains
```

POST ./research

Dieser Endpunkt wird bei der Recherche nach Wappen angesprochen. Die Suchparameter werden im Request-Body übergeben. Dabei handelt es sich um den Namen des Wappens, die Lokation, eine Liste einzelner Begriffe, die in den Begriffsketten vorkommen sollen (entspricht einer Stichwortsuche), sowie eine Liste von UUID-Listen. Die UUIDs gehören zu Begriffen in der Datenbank und eine UUID-Liste entspricht einer vom Nutzer eingegebenen (Teil-)Begriffskette, die die gesuchten Wappen aufweisen sollen.
Der Endpunkt ruft zwei aufeinanderfolgende Anfragen auf. Die erste gibt für jede vom Nutzer geforderte (Teil-)Begriffskette eine Liste mit Begriffsketten-UUIDs aus der Datenbank zurück, die die Begriffskette des Nutzers beinhalten.

Listing 5.5: Erste Anfrage zu `/research`

```
1  // $termUUIDs ist die Liste von UUID-Listen aus dem Request
2  UNWIND $termUUIDs AS chain
3  CALL {
4       WITH chain
5       MATCH (t:Term)
6       WHERE t.uuid IN chain
7       RETURN collect(t) AS terms
8    }
9  MATCH (ch:Chain)
10 WHERE ALL(term IN terms WHERE (ch)-[:CONTAINS_TERM]->(term))
11 RETURN chain, collect(ch.uuid) AS UUIDs
```

Die zweite Anfrage gibt dann alle Wappen aus der Datenbank aus, deren Name und Lokation zu den angefragten Werten aus dem Request passen, die jede der im Request angefragten Begriffsketten beinhalten (erfolgt über das Ergebnis der ersten Anfrage), und in deren Begriffsketten mindestens einer der angefragten einzelnen Begriffe enthalten ist.

Listing 5.6: Zweite Anfrage zu `/research`

```
1  // $name, $location sind Name und Lokation aus dem Request
2  // $singleTerms ist die Liste mit Einzelbegriffen aus dem Request
3  // $chainListsToSatisfy ist das Ergebnis der ersten Anfrage: eine
   Liste von Listen mit Begriffsketten-UUIDs
4
5  MATCH (coa:CoA)
6  WHERE toLower(coa.name) CONTAINS $name
7  AND toLower(coa.location) CONTAINS $location
8  AND ALL(chainList IN $chainListsToSatisfy WHERE ANY(chain IN
   chainList WHERE (coa)-[:HAS_CHAIN]->(:Chain {uuid: chain})))
9  CALL apoc.when(
10    $singleTerms = [''],
11    'RETURN DISTINCT coa',
12    'MATCH (coa)-[:HAS_CHAIN]->(:Chain)-[:CONTAINS_TERM]->(t:Term)
13    WHERE ANY(term in $singleTerms WHERE toLower(t.name) CONTAINS
      term)
14    RETURN DISTINCT coa',
```

```
15      {coa: coa, singleTerms: $singleTerms}
16  ) YIELD value
17  RETURN value.coa.uuid AS UUID
```

POST ./terms/upsertTerm

Dieser Endpunkt wird zum Anlegen oder Verändern von Begriffen verwendet. Im Request-Body werden eine Begriffs-UUID, die UUID eines Vorgängerbegriffes und die Daten des anzulegenden/zu verändernden Begriffes erwartet. Ist die Begriffs-UUID leer, wird eine neuer Begriffsknoten mit einer zufällig erzeugten UUID angelegt, anderenfalls die Properties des bestehenden Knotens verändert. Ist im ersten Fall die Vorgänger-UUID ebenfalls leer, wird der neue Knoten ohne Vorgänger angelegt, anderenfalls mit dem entsprechenden Begriffsknoten als Vorgänger.

Listing 5.7: Anfragen zu `/terms/upsertTerm`

```
1   // $attributes enthält in jedem Fall die Daten des
    neuen/veränderten Begriffs
2
3   // Anfrage zur Neuanlage des Begriffes ohne Vorgänger
4   CREATE (t:Term {uuid: randomUUID()})
5   SET t += $attributes
6   RETURN t.uuid AS UUID
7
8   // Anfrage zur Neuanlage des Begriffes mit Vorgänger
9   // $parent ist die UUID des Vorgängerbegriffes
10  MATCH (parent:Term)
11  WHERE parent.uuid = $parent
12  CREATE (parent)-[:NEXT_TERM]->(t:Term {uuid: randomUUID()})
13  SET t += $attributes
14  RETURN t.uuid AS UUID
15
16  // Anfrage zum Verändern eines bestehenden Begriffes
17  // $uuid ist die UUID des zu verändernden Begriffes
18  MATCH (t:Term)
19  WHERE t.uuid = $uuid
20  SET t += $attributes
21  RETURN t.uuid AS UUID
```

GET ./terms/deleteTerm

Mit Aufruf dieses Endpunkts kann ein Begriffsknoten wieder gelöscht werden. Die UUID des zu löschenden Begriffs wird im Query-String übergeben. In der Anfrage werden zunächst die Anzahl an Begriffsketten, die den Begriff enthalten, und die Anzahl an Nachfolgerbegriffen ermittelt. Nur wenn beide Werte Null sind, wird der Begriff auch tatsächlich gelöscht. Anderenfalls werden die ermittelten Werte in der Response an den Client zurückgegeben.

Listing 5.8: Anfrage zu `/terms/deleteTerm`

```
1   // $termUUID ist die UUID des zu löschenden Begriffs aus dem
    Request
2   MATCH (term:Term)
3   WHERE term.uuid = $termUUID
4   WITH term, size((term)<-[:CONTAINS_TERM]-()) AS numberChains,
    size((term)-[:NEXT_TERM]->()) AS numberTerms
```

```
5   CALL apoc.do.when(
6      numberChains = 0 AND numberTerms = 0,
7      'DETACH DELETE term RETURN numberChains, numberTerms',
8      'RETURN numberChains, numberTerms',
9      {term: term, numberChains: numberChains, numberTerms:
       numberTerms}"
10  ) YIELD value
11  RETURN value.numberChains AS Chains, value.numberTerms AS Terms
```

GET ./terms/addTermRelationship

Bei diesem Endpunkt wird lediglich eine :NEXT_TERM-Relationship zwischen zwei bereits existierenden Begriffsknoten neu angelegt. Die UUIDs der beiden Knoten stammen aus dem Query-String.

Listing 5.9: Anfrage zu /terms/addTermRelationship

```
1   // $parent, $child sind die UUIDs des Start-/Endknotens
2   MATCH (parent:Term), (child:Term)
3   WHERE parent.uuid = $parent AND child.uuid = $child
4   CREATE (parent)-[:NEXT_TERM]->(child)
```

GET ./terms/removeTermRelationship

Hierbei handelt es sich um die inverse Operation zu dem zuvor genannten Endpunkt. Eine bestehende :NEXT_TERM-Relationship zwischen zwei Begriffsknoten wird wieder entfernt. Die UUIDs des Start- und Endknotens werden im Query-String übergeben. Die Relationship wird nur unter der Bedingung gelöscht, dass es keine Begriffskette gibt, die beide Begriffe enthält.

Listing 5.10: Anfrage zu /terms/removeTermRelationship

```
1   // $parent, $child sind die UUIDs des Start-/Endknotens
2   MATCH (parent:Term)-[r:NEXT_TERM]->(child:Term)
3   WHERE parent.uuid = $parent AND child.uuid = $child
4   AND NOT
    (parent)<-[:CONTAINS_TERM]-(:Chain)-[:CONTAINS_TERM]->(child)
5   DELETE r
6   RETURN count(r) AS NumberDeleted
```

POST ./coa/upsertCoA

Dieser Endpunkt dient zur Neuanlage eines Wappens mit Begriffsketten sowie zum Ändern der Daten oder Begriffsketten eines bestehenden Wappens. Im Request-Body werden die UUID des Wappens, dessen Daten wie zum Beispiel der Name und eine Liste mit Begriffsketten übergeben. Die einzelnen Begriffsketten werden wiederum als Liste von Begriffs-UUIDs repräsentiert. Vor der Anfrage an die Datenbank werden zunächst noch die übergebenen Begriffsketten mit ihren Begriffen modifiziert, indem jedem Begriff und jeder Begriffskette die Position in der jeweiligen Liste zugeordnet wird. In der Anfrage wird diese Position dann als order-Property der Relationships verwendet.
Abhängig davon, ob die Wappen-UUID angegeben leer ist oder nicht, wird eine von zwei Anfragen ausgeführt. Ist sie leer, werden in der entsprechenden Anfrage schrittweise ein neues Wappen, die zugehörigen Begriffsketten und die Relationships zu den Begriffen angelegt.

Listing 5.11: Anfrage zur Neuanlage eines Wappens in `/coa/upsertCoA`

```
1  // $attributes: Metadaten des Wappens (Name, Lokation...)
2  // $chains: Liste von anzulegenden Begriffsketten
3  CREATE (coa:CoA {uuid: randomUUID()})
4  SET coa += $attributes
5  WITH coa
6  UNWIND $chains AS chain
7  CREATE (coa)-[:HAS_CHAIN {order: chain.index}]->(ch:Chain {uuid:
   randomUUID()})
8  WITH coa, chain, ch
9  UNWIND chain.containedTerms AS term
10 MATCH (t:Term)
11 WHERE t.uuid = term.term
12 CREATE (ch)-[:CONTAINS_TERM {order:term.index}]->(t)
13 RETURN DISTINCT coa.UUID AS UUID
```

Ist die Wappen-UUID im Request nicht leer, wird das so angegebene Wappen modifiziert. Neben den Metadaten betrifft das auch die Begriffsketten. Neu angelegte Begriffketten werden hinzugefügt und die Reihenfolge entsprechend angepasst. Schließlich werden vom Nutzer entfernte Begriffsketten aus der Datenbank gelöscht.

Listing 5.12: Anfrage zur Modifizierung eines bestehenden Wappens in `/coa/upsertCoA`

```
1  // $attributes: Metadaten des Wappens (Name, Lokation...)
2  // $chains: Liste von anzulegenden Begriffsketten
3  MATCH (coa:CoA)
4  WHERE coa.uuid = $uuid
5  SET coa += $attributes
6  WITH coa
7  UNWIND $chains AS chain
8  MERGE (coa)-[hcRel:HAS_CHAIN]->(ch:Chain {uuid: chain.uuid})
9  ON CREATE
10   SET ch.uuid = randomUUID()
11 SET hcRel.order = chain.index
12 WITH coa, chain, ch
13 UNWIND chain.containedTerms AS term
14 MATCH (t:Term)
15 WHERE t.uuid = term.term
16 MERGE (ch)-[ctRel:CONTAINS_TERM]->(t)
17 SET ctRel.order = term.index
18 WITH coa, collect(ch.uuid) AS chainUUIDs
19 MATCH (coa)-[:HAS_CHAIN]->(oldChain:Chain)
20 WHERE NOT oldChain.uuid IN chainUUIDs
21 DETACH DELETE oldChain
22 RETURN DISTINCT coa.uuid AS UUID
```

`GET ./coa/deleteCoA`

Die Aufgabe dieses Endpunkts ist das Löschen eines Wappens. Dazu wird im Query-String die UUID des Knotens übergeben. Anschließend werden in der Anfrage das Wappen und seine Begriffsketten gelöscht.

Listing 5.13: Anfrage in `/coa/deleteCoA`

```
1  // $coa: UUID des zu löschenden Wappens
2  MATCH (coa:CoA)
3  WHERE coa.uuid = $coa
4  MATCH (chain:Chain)<-[:HAS_CHAIN]-(coa)
5  DETACH DELETE chain
6  DETACH DELETE coa
```

5.2.3 Angular-Frontend

Grundsätzlich implementiert das Angular-Frontend die drei in Kapitel 4.3.1 beschriebenen Funktionseinheiten des Begriffseditors, der Wappenerfassung und der Wappenrecherche. Die einzelnen Einheiten sind jeweils in einer Angular-Komponente umgesetzt, die als Container fungiert, und über eine im `RouterModule` definierte URL angesteuert werden kann. Eine weitere Komponente dient als Startseite, die der Nutzer beim Aufruf der Webseite zuerst sieht. Darauf sollte dem Nutzer nach Möglichkeit die Funktionsweise des Systems erklärt werden oder relevante Dokumente oder andere Webseiten verlinkt sein. Zum Abgabezeitpunkt der vorliegenden Arbeit wurde diese „Startkomponente“ allerdings noch nicht implementiert und hat keinen Inhalt. Abbildung 5.12 gibt einen Überblick über die im `RouterModule` definierten URLs.
Zur Navigation zwischen den einzelnen Komponenten der Funktionseinheiten wird das Material-Design-Element *Sidenav*[7] verwendet. Dadurch ist auf der linken Seite des User Interfaces ein Overlay ein- und ausklappbar, das Verlinkungen zu den gerouteten Komponenten beinhaltet. Das Anklicken eines Links rendert dann den Inhalt der entsprechenden Komponente im dafür vorgesehenen Container des Sidenav-Elements. Abbildung 5.13 zeigt das ausgeklappte Sidenav-Element mit den vorhandenen Links. Im Hintergrund ist die `coa-list`-Komponente zu sehen.
Die Kommunikation zwischen dem Frontend und der Flask-API wird mittels Angular-*Services* durchgeführt, die über Angulars Dependency-Injection-Framework auf die `HttpClient`-Klasse zugreifen und darüber Requests senden können. Die Services selbst werden dann ebenfalls über Dependency Injection in Components injiziert, sodass diese die Funktionen des Services aufrufen können. Ein Beispiel für so eine Serviceklasse ist in Abbildung 5.14 zu finden.
In den folgenden Abschnitten soll jetzt auf die Implementierung der einzelnen Funktionseinheiten Begriffseditor, Wappenerfassung und Wappenrecherche eingegangen werden.

Begriffseditor

Der Begriffseditor vereint in sich die meiste Funktionalität. Begriffe können angelegt, bearbeitet und wieder gelöscht werden. Zusätzlich gibt es die Möglichkeit, Relationships zwischen zwei bereits bestehenden Begriffen anzulegen oder zu entfernen. Um bei diesen Operationen eine schnelle Navigation im Begriffsgraphen zu ermöglichen und schnell von Begriff zu Begriff springen zu können, werden die zuletzt ausgewählten Begriffe in einer Liste gespeichert und können daraus bei Bedarf schnell wieder ausgewählt werden.
Die Komponente des Begriffseditors (`termeditor-container.component`) besteht wiederum aus anderen Komponenten, die jeweils einen Teil der Funktionalität übernehmen. Deren Abgrenzungen sind in Abbildung 5.15 zu sehen. Die `termeditor-topbar.component`-Komponente zeigt immer die Daten des aktuell ausgewählten Begriffes an, also Name, Bemerkung, Synonyme und ob es ein Sammelbegriff ist. Weiterhin beinhaltet sie eine Reihe von Buttons, die die folgenden Effekte haben:

- **Zurück zum vorigen Begriff**: Wählt den zuletzt ausgewählten Begriff aus.

[7] `https://material.angular.io/components/sidenav/overview`, zuletzt abgerufen am 27.08.2021

```
1  // app.module.ts
2
3  @NgModule({
4    //...
5    imports: [
6      // ...
7      RouterModule.forRoot([
8        // Startkomponente
9        {path: "", component: DummyComponent},
10       // Komponente für Begriffseditor (Begriffseditor)
11       {path: "termeditor", component:
         TermeditorContainerComponent},
12       // Komponente mit Liste aller erfassten Wappen
         (Wappenerfassung)
13       {path: "coa-list", component: CoaListComponent},
14       // Komponente zur Neuanlage/Modifizierung eines Wappens
         (Wappenerfassung)
15       {path: "upsert-coa", component: UpsertCoaComponent},
16       {path: "upsert-coa/:coaUUID", component:
         UpsertCoaComponent},
17       // Komponente zur Recherche nach Wappen (Wappenrecherche)
18       {path: "coa-research", component: ResearchCoaComponent},
19       {path: "coa-details/:coaUUID", component:
         CoaDetailsComponent}
20     ])
21     //...
22   ],
23   //...
24 })
```

Abbildung 5.12: Frontend: Übersicht über die verfügbaren URLs mit den zugehörigen Komponenten

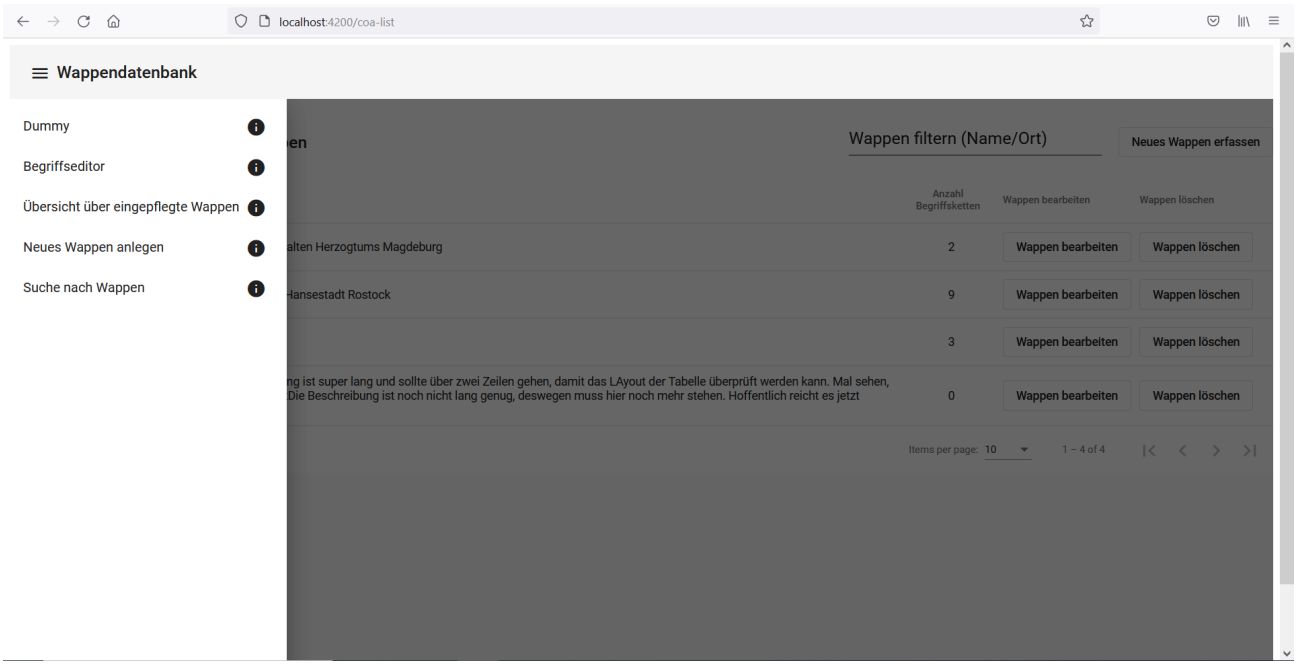


Abbildung 5.13: Frontend: Ansicht des ausgeklappten Sidenav-Elements

```
1  // termeditor-backend-service.service.ts
2  @Injectable({
3    providedIn: 'root'
4  })
5  export class TermeditorBackendServiceService {
6    // Serveradresse wird aus Konfigurationsdateien gelesen
7    private BASE_URL = environment.baseURL;
8
9    // HttpClient Instanz wird im Konstruktor über Dependency
     Injection injiziert
10   constructor(private http: HttpClient) { }
11
12   // Funktion sendet Request an die entsprechende Server-URL
13   getAllTerms(): Observable<Term[]> {
14     return this.http.get<Term[]>(this.BASE_URL +
       "termeditor/allTerms");
15   }
16 }
```

Abbildung 5.14: Frontend: Beispiel für eine Service-Klasse

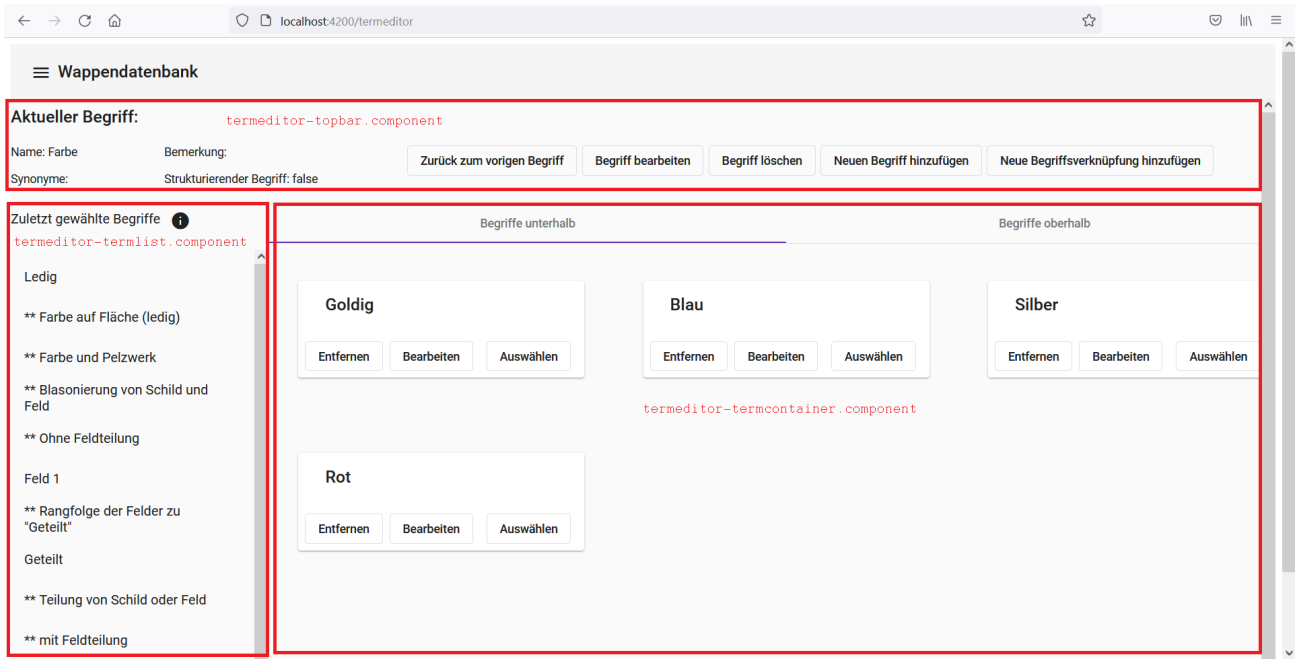


Abbildung 5.15: Frontend: Abgrenzungen der Komponenten im Begriffseditor

- **Begriff bearbeiten**: Öffnet einen Dialog, in dem die Daten des aktuellen Begriffs bearbeitet werden können.
- **Begriff löschen**: Löscht den aktuell ausgewählten Begriff.
- **Neuen Begriff hinzufügen**: Öffnet einen Dialog, in dem ein neuer Begriff als Nachfolger des aktuellen Begriffes angelegt werden kann.
- **Neue Begriffsverknüpfung hinzufügen**: Öffnet einen Dialog, in dem aus einer Liste aller Begriffe ein Begriff ausgewählt werden kann, der als Vorgänger oder Nachfolger des aktuellen Begriffes definiert werden soll.

Die `termeditor-termlist.component`-Komponente beinhaltet lediglich eine Liste der bisher ausgewählten Begriffe, die durch Auswahl des jeweils nächsten Begriffes kontinuierlich erweitert wird. Ein Klick auf eines der Listenelemente wählt diesen Begriff als aktuellen Begriff aus und entfernt die Listenelemente, die nach dem Ausgewählten kommen, aus der Liste.
Der in Abbildung 5.15 als `termeditor-termcontainer.component` bezeichnete Bereich beinhaltet zunächst einmal zwei *Tab*-Material-Design-Elemente[8], die dann jeweils eine `termeditor-termcontainer.component`-Komponente beinhalten. Der erste Tab stellt die Nachfolger-, der zweite die Vorgängerbegriffe des aktuellen Begriffs in einer Liste dar. Diese Listen bestehen wiederum aus `term.component`-Komponenten, die ein *Card*-Material-Design-Element[9] beinhalten. Dieses zeigt die Daten des jeweiligen Begriffes an und bietet Optionen, diese Daten zu bearbeiten, den Begriff als aktuellen auszuwählen oder die Relationship zwischen dem aktuell im Begriffseditor ausgewählten Begriff und dem Begriff der `term.component`-Komponente zu löschen. Der beschriebene Aufbau des Begriffseditors wird in Abbildung 5.16 in einem vereinfachten visualisierten DOM nochmal veranschaulicht.

[8] `https://material.angular.io/components/tabs/overview`, zuletzt abgerufen am 27.08.2021
[9] `https://material.angular.io/components/card/overview`, zuletzt abgerufen am 27.08.2021

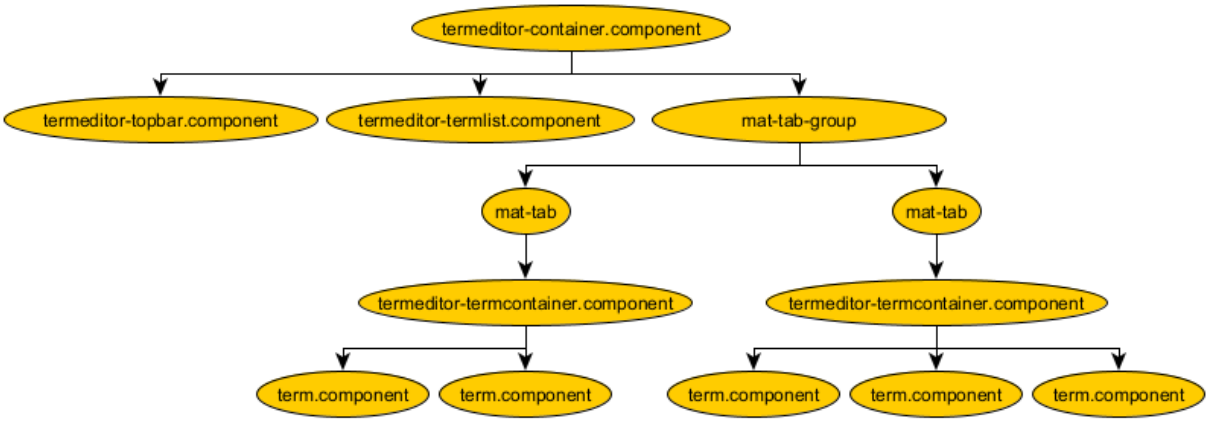


Abbildung 5.16: Frontend: Vereinfachtes DOM der Begriffseditor-Komponente

Wappenerfassung

Die Einheit der Wappenerfassung besteht aus zwei Komponenten, eine zur Übersicht über alle Wappen (`coa-list.component`) und eine zum Bearbeiten beziehungsweise zur Neuanlage eines einzelnen Wappens (`upsert-coa.component`). Beide sind über Links in der seitlichen Navigationsleiste zu erreichen.
Die `coa-list.component`-Komponente besteht im Wesentlichen aus einem *Table*-Material-Design-Element[10]. In dessen Spalten werden Name, Lokation, Beschreibung und die Anzahl der Begriffsketten des Wappens angezeigt. In weiteren Spalten befinden sich Buttons zum Bearbeiten und Löschen der Wappen, wobei der Button zum Bearbeiten auf die genannte zweite Komponente weiterleitet. Über ein Textfeld oberhalb der Tabelle können die angezeigten Wappen gefiltert werden. Abbildung 5.17 zeigt einen Screenshot dieser beschriebenen Wappenliste.
Die `upsert-coa.component`-Komponente dient zur Neuanlage und zum Bearbeiten von bestehenden Wappen. Im zweiten Fall wird die Komponente ausgehend von der zuvor beschriebenen `coa-list.component`-Komponente aufgerufen. Dabei wird die UUID des betreffenden Wappens im Query-String mit an die `upsert-coa.component`-Komponente übergeben und die Daten in die Formularfelder übernommen. Bei der Neuanlage bleiben alle Felder zunächst leer. Neben der Eingabe der Metadaten Name, Lokation und Beschreibung können ebenfalls Begriffsketten angelegt werden. Dazu öffnet sich bei Klick auf den entsprechenden Button ein Dialog, in dem schrittweise die Begriffe der Kette ausgewählt werden können. Anschließend besteht noch die Möglichkeit, die Reihenfolge der angelegten Begriffsketten zu verändern.

Wappenrecherche

Die Wappenrecherche ist im Wesentlichen eine Kombination der Elemente beider Komponenten der Wappenerfassung. In einer Suchmaske kann der Nutzer Suchbegriffe für Name, Lokation und einzelne Begriffe eingeben. Darüber hinaus können über einen Dialog Begriffsketten definiert werden, die die gesuchten Wappen haben müssen. Dabei muss nicht zwangsläufig eine vollständige Begriffskette angegeben werden. Es besteht auch die Möglichkeit, nur Teile einer Kette anzugeben. Insgesamt ist dieser Teil ähnlich aufgebaut wie die `upsert-coa.component`-Komponente bei der Wappenerfassung.
Nachdem der Nutzer alle Suchparameter eingegeben und die Suche gestartet hat, wird das Ergebnis in einer Tabelle ähnlich wie in der `coa-list.component`-Komponente angezeigt.

[10] `https://material.angular.io/components/table/overview`, zuletzt abgerufen am 27.08.2021

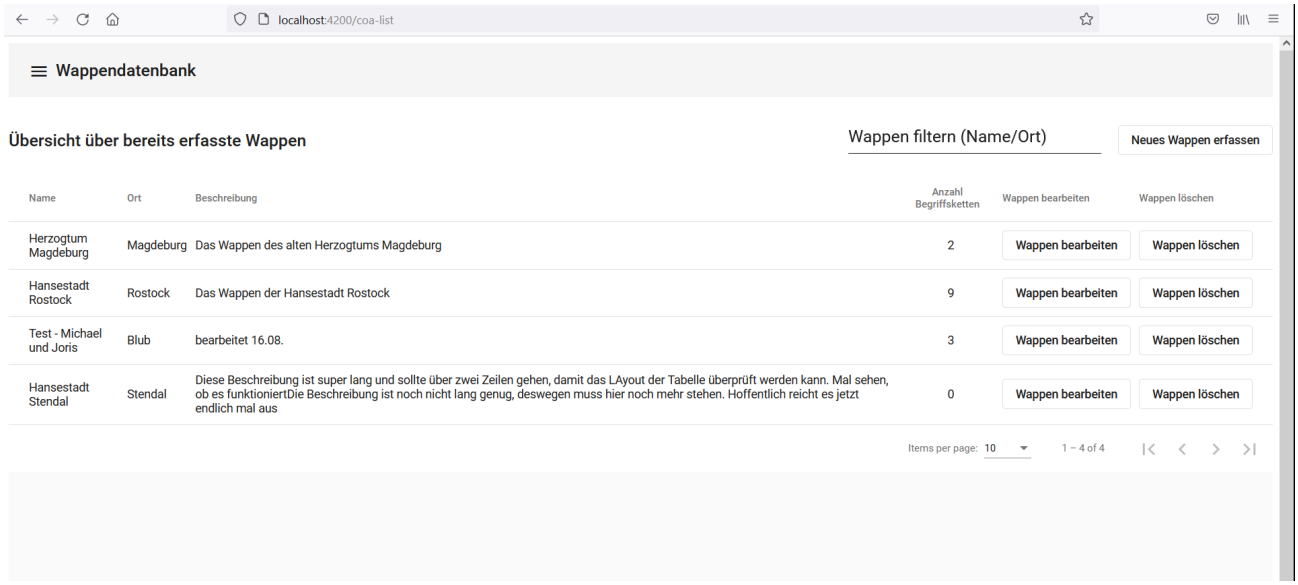


Abbildung 5.17: Frontend: Screenshot der Übersicht über alle Wappen

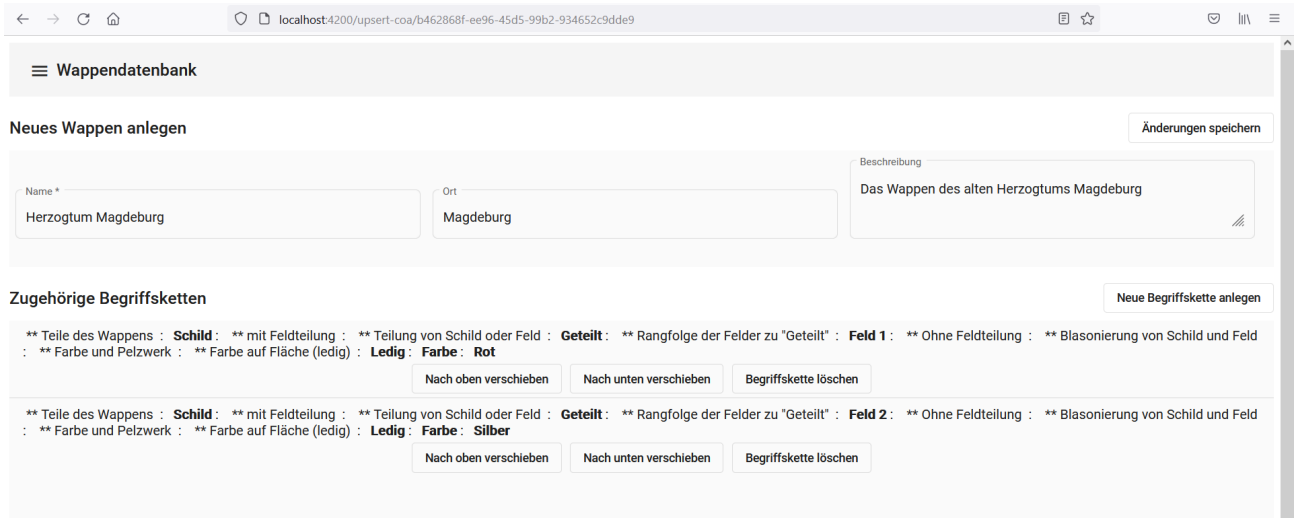


Abbildung 5.18: Frontend: Ansicht zur Bearbeitung eines Wappens

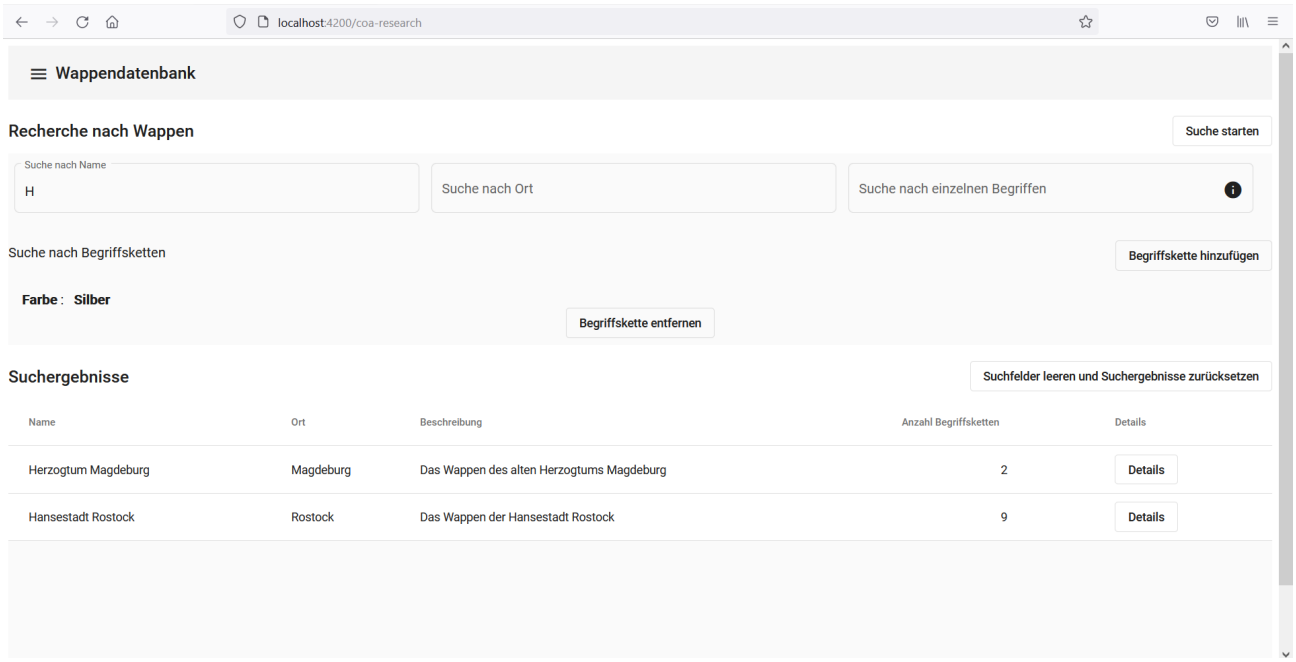


Abbildung 5.19: Frontend: Wappenrecherche

Ein Screenshot der Suche ist in Abbildung 5.19 zu sehen. Es wird nach Wappen gesucht, deren Name den Buchstaben „h“ beinhaltet und bei denen in mindestens einer Begriffsketten die Teilkette „Farbe: Silber“ vorhanden ist.

Kapitel 6

Zusammenfassung und Ausblick

Nach einem kurzen Überblick über die Heraldik, das Semantic Web und das Property-Graph-Modell in Kapitel 2 wurden in Kapitel 3 dieser Arbeit einige Ideen zur digitalen Speicherung von Wappen vorgestellt. Im Gegensatz zur bisherigen Arbeitsweise, die auf der (im schlimmsten Fall linearen) Suche in gedruckten Wappensammlungen beruhte, in denen die Wappen in Volltext blasoniert und nach Namen sortiert sind, bieten die vorgestellten Ideen einerseits eine Formalisierung der Blasonierungssprache und andererseits die Möglichkeit, inhaltliche Gesichtspunkte bei der Recherche nach Wappen einzubinden. Hierbei ist insbesondere der in Unterkapitel 3.2 diskutierte Ansatz von Torsten Hiltmann und Thomas Riechert zu erwähnen, in dem die Idee einer Ontologie zur Wappenbeschreibung im Rahmen des Semantic Web aufgeworfen wird. Wappen würden nach dieser Methode als RDF-Graph abgespeichert werden. Das Problem hierbei ist, dass die Blasonierungssprache sehr weitläufig ist und damit die Definiton einer umfassenden Ontologie sehr komplex, sodass aus der Idee bisher noch kein nutzbares Ergebnis hervorgegangen ist.
Einen ähnlichen Ansatz verfolgt Michael Voß mit seiner Idee der Begriffsketten, die in Unterkapitel 3.3 erläutert wird. Im Wesentlichen wird dabei die gängige Volltextblasonierung auf eine Hintereinanderreihung (die „Kette“) von heraldischen Begriffen reduziert. Die Begriffe und ihre Verknüpfungen untereinander, welcher Begriff vor und nach welchem anderen Begriff stehen kann, bilden dabei einen Graphen, den sogenannten Begriffsgraphen. Dieser Ansatz wurde bisher auf Voß' Eigeninitiative in einer Webanwendung implementiert und von ihm zur Wappenerfassung verwendet.
Ziel dieser Arbeit war, den Prototyp einer neuen Webanwendung zu entwickeln, der das Konzept der Begriffsketten aufgreift und für eine breite Nutzergruppe zugänglich macht. Als Technologien und Frameworks werden eine Neo4j-Graphdatenbank, das Web-Framework Angular für das Frontend und das Python-Framework Flask als Schnittstelle zur Kommunikation zwischen Frontend und Datenbank verwendet. Der Prototyp umfasst die drei Funktionseinheiten des Begriffseditors, der Wappenerfassung und der Wappenrecherche. Einige für die Nutzung der neuen Anwendung unter realen Bedingungen essenzielle Features, beispielsweise das in Unterkapitel 4.1 beschriebene Rollenmodell, konnten dabei mit den zeitlichen und personellen Ressourcen dieser Bachelorarbeit leider nicht umgesetzt werden. Weitere Details zum Konzept und zur Implementierung des Prototypen finden sich in den Kapiteln 4 und 5.

Insgesamt lässt sich feststellen, dass die Digitalisierung der heraldischen Arbeit unter dem Stichwort der *Digital Humanities* unerlässlich ist, um auch im 21. Jahrhundert die Bedeutung der Heraldik als Hilfswissenschaft der Geschichtsforschung zu erhalten. Einerseits benötigt es dazu Systeme, um große Wappensammlungen verwalten zu können, andererseits aber erst mal

ein etabliertes Datenformat, um Wappen mit ihren Metadaten und Blasonierungen speichern zu können. Zum gegenwärtigen Zeitpunkt ist ein solches allerdings nicht bekannt.
Eine Idee für ein System zur Verwaltung von Wappen bietet das beschriebene Format der Begriffsketten von Michael Voß. Der im Rahmen dieser Arbeit entwickelte Prototyp stellt den Anfang der Entwicklung einer Anwendung dar, die die Erfassung und Bereitstellung einer Wappensammlung durch Zusammenarbeit verschiedener Nutzer im Sinne eines Crowdsourcing-Projekts als zentrales Ziel hat. Bevor diese Anwendung allerdings wie angedacht im Rahmen des Ortschronikenportals Mecklenburg-Vorpommern zum Einsatz gebracht werden kann, ist die Umsetzung einiger zentraler Features vonnöten, namentlich die Implementierung des Rollenmodells, die Einbindung von Bildern für Wappen und Begriffe sowie die Nutzung von Responsive Design zur Anpassung der Benutzeroberfläche für mobile Endgeräte. Erst dann kann sich bei Verwendung durch eine hinreichend große Nutzergruppe zeigen, ob die Begriffsketten das Potential haben, sich als Format zur Wappenbeschreibung großflächig durchzusetzen.

Literaturverzeichnis

[BBUW98] Balke, Wolf-Tilo, Thomas Birke, Katharina Urch und Matthias Wagner: *Das HERON-Projekt - Ein Zwischenbericht.* Oktober 1998.

[BH17] Biewer, Ludwig und Eckart Henning: *Wappen: Handbuch der Heraldik, als „Wappenfibel“ begründet von Adolf Matthias Hildebrandt.* Böhlau Verlag, 20 Auflage, 2017.

[BLHL01] Berners-Lee, Tim, James Hendler und Ora Lassila: *The Semantic Web: A New Form of Web Content That is Meaningful to Computers Will Unleash a Revolution of New Possibilities.* ScientificAmerican.com, 2001.

[Cle] Clemmensen, Steen: *Ordinary of Medieval Armorials 2.0.* 2017, zuletzt aktualisiert 2021, `http://www.armorial.dk/`, zuletzt abgerufen am 29.06.2021.

[Fla] *Foreword.* `https://flask.palletsprojects.com/en/2.0x/foreword`, zuletzt abgerufen am 24.08.2021.

[Goo] Google: *What is Angular?* `https://angular.io/guide/what-is-angular`, zuletzt abgerufen am 26.08.2021.

[Hit21] Hitzler, Pascal: *A Review of the Semantic Web Field.* Communications of the ACM, 64(2):76 – 83, Januar 2021.

[Hol10] Holley, Rose: *Crowdsourcing: How and why should libraries do it?* D-Lib Magazine, 16(3/4 Ma), 2010.

[HR20] Hiltmann, Torsten und Thomas Riechert: *Digital Heraldry. The State of the Art and New Approaches Based on Semantic Web Technologies.* In: Balouzat-Loubet, Christelle (Herausgeber): *Digitizing Medieval Sources - L'édition en ligne de documents d'archives médiévaux: Challenges and Methodologies - Enjeux, méthodologie et défis.* Brepols Publishers, Turnhout, Belgien, 2020.

[Jan16] Janndis, Fotis: *Digital Humanities.* `https://gi.de/informatiklexikon/digital-humanities`, zuletzt abgerufen am 16.05.2021, 2016.

[Neo] Neo4j: *Graph Modeling Guidelines.* `https://neo4j.com/developer/guide-data-modeling/`, zuletzt abgerufen am 26.08.2021.

[Sch11] Schütt, Hans-Heinz: *Auf Schild und Flagge: Die Wappen und Flaggen des Landes Mecklenburg-Vorpommern und seiner Kommunen.* produktionsbüro TINUS, Schwerin, 2011.

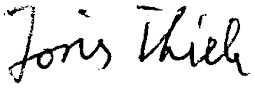
[SSH18] Saake, Gunter, Kai-Uwe Sattler und Andreas Heuer: *Datenbanken: Konzepte und Sprachen.* mitp Verlag, 6 Auflage, 2018.

[UBK99] Urch, Katharina, Wolf-Tilo Balke und Werner Kiessling: *Erfahrungen in der Digitalisierung und Erschließung einer historischen Wappensammlung*. ABI-Technik, 20, März 1999.

[Voß21] Voss, Michael: *Kunstgut-Inventarisierung und Heraldik: zwei Fachgebiete, ein Datenbankprogramm; netzwerkorientiertes Datennmodell in Anlehnung an Semantic-Web-Technologien*. Eigenverlag, 2021.

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.



Rostock, den 29. August 2021