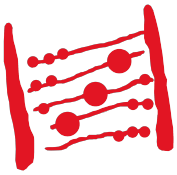

Fachbereich 3: Mathematik und Informatik

Bachelorarbeit

Entwicklung und Implementierung einer Webplattform für ein inklusives Wörterbuch mit OER-Materialien

Development and implementation of an online-platform for an inclusive dictionary with OER materials

Dennis Schürholz
schuerholz@uni-bremen.de
Matrikel-Nr. 400 623 1

19. Juli 2019

1. Gutachter: Dr. Yıldıray Oğurol
2. Gutachter: Prof. Dr. Rainer Koschke

Dennis Schürholz, schuerholz@uni-bremen.de

Entwicklung und Implementierung einer Webplattform für ein inklusives Wörterbuch mit OER-Materialien

Development and implementation of an online-platform for an inclusive dictionary with OER materials

Bachelorarbeit, Fachbereich 3: Mathematik und Informatik

Universität Bremen, Juli 2019

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Bremen, den 19. Juli 2019

_____

Dennis Schürholz

Danksagung

An dieser Stelle möchte ich mich all denjenigen bedanken, die mich während der Anfertigung dieser Bachelorarbeit unterstützt und motiviert haben.

Zuerst bedanke ich mich bei Dr. Yıldıray Oğurol, der meine Bachelorarbeit betreut und begutachtet hat. Für die ausführliche konstruktive Kritik und vielen hilfreichen Tipps in mehreren sehr produktiven Gesprächen möchte ich mich herzlich bedanken. Zusätzlich möchte ich Prof. Dr. Rainer Koschke für die Übernahme des Zweitgutachten und die Beratung und Beharrlichkeit vor Beginn der Bachelorarbeit bedanken.

Im Weiteren möchte ich Prof. Dr. Frank J. Müller danken für die Möglichkeit diese Projektarbeit im Kontext seines Arbeitsgebietes durchzuführen und weiterhin für die aufgebrachte Geduld und das kontinuierliche Feedback während der Umsetzung der Arbeit.

Ebenfalls bedanke ich mich bei meinen Kommilitonen Julian Gebken und Johannes Ganser für die Gespräche zu meiner Arbeit und den daraus resultierenden Ideen und Anregungen.

Weiterhin bedanke ich mich bei meinen Kommilitonen Enno Gerhard und Bino Nolting, sowie meiner Mutter Hannelore Schürholz für das Korrekturlesen dieser Bachelorarbeit.

Außerdem bedanke ich mich bei Jessica Winter und Mareen Koops für die emotionale und mentale Unterstützung sowie die entgegengebrachte Geduld für die Anfertigung meiner Bachelorarbeit und die Beendigung dieses Studienabschnittes.

Ich möchte mich abschließend bei meinen nicht explizit genannten Familienmitgliedern und meinen weiteren engen Freunden für die Unterstützung während des Studiums und insbesondere während der Zeit der Entwicklung meiner Bachelorarbeit bedanken.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Zu den Kernkompetenzen von Lehrkräften gehört die Aufbereitung des Lehrstoffes z. B. durch das Entwerfen und Weiterverarbeiten von Unterrichtsmaterialien (Brause und Schulz 2017). Als Unterstützung sind die Lehrkräfte daher auf strukturierte, mit Metadaten angereicherte und untereinander verknüpfte, aufbereitete Grundmaterialien angewiesen, um sie in diesem Arbeitsalltag produktiv zu unterstützen. In Bezug auf den Deutschunterricht in Grundschulen werden Arbeitsmaterialien zu diversen Unterrichtsfeldern wie Reimwörtern, Themengruppen oder Rechtschreibphänomenen benötigt. Zur Erstellung von eigenen aufbereiteten Materialien zu solchen Themenkomplexen sind strukturierte Ausgangsmaterialien ein wichtiger Grundsatz. Diese Problemstellung greift die vorliegende Bachelorarbeit auf und entwickelt ein webbasiertes System zur Sammlung und Bereitstellung eben solcher angereicherter Datensätze. In der Anforderungserhebung und dem grundsätzlichen Aufbau des zu entwickelnden Systems wird mit Prof. Dr. Frank J. Müller[1] zusammengearbeitet, welcher damit als Projektpartner bzw. Kunde agiert. In seinem Projekt »#UnsereWörter« werden »inklusive freie Bildungsmaterialien zum interessengeleiteten Schriftspracherwerb [entstehen]. Ziel ist es allen Schülerinnen und Schülern, ausgehend von ihrer Lebenswelt, Materialien zum Schriftspracherwerb zur Verfügung zu stellen. Diese sollen durch die Lehrkräfte flexibel an die Bedürfnisse der Kinder und Jugendlichen angepasst werden können« (Müller 2019b).

Im Rahmen dieses Projektes wird eine mit (Meta-)Daten angereicherte Wortschatzsammlung benötigt. Diese soll stetig erweiterbar und ohne Barrieren zugreifbar sein und daher als Webplattform frei verfügbar sein. Dieses System stellt ein digitales inklusives Wörterbuch zur Schaffung von Open Educational Ressources (OER)-Materialien unter Einbeziehung von eben solchen Inhalten dar. Um in der Projektumsetzung im Gedanken von OER zu handeln wird auf auch hier auf offene Standards und Komponenten gesetzt.

Ziel der Arbeit ist somit ein Softwaresystem als Werkzeug zu schaffen, welches die kreative Arbeit der Lehrkräfte unterstützt und so die Vielfalt von Lehr- und Lernmaterialien vergrößert. Dabei ist es als Ergänzung zu bestehenden Datenbanken mit Arbeitsmaterialien gedacht um so auch den Schaffungsprozess dieser Materialien zu begleiten. Weiterhin soll im Rahmen der Bachelorarbeit eine solide Basis geschaffen

[1] Prof. Dr. Frank J. Müller ist Juniorprofessor für inklusive Pädagogik mit den Schwerpunkten Geistige Entwicklung und Lernen am Fachbereich 12 der Universität Bremen

werden, auf deren Grundlage die Nutzung und der Einsatz des Systems getestet und untersucht werden kann. Aufbauend auf dieser Grundlage sollen langfristig auch Erweiterungen und Ergänzungen bzw. weitere angegliederte Systeme konzipiert und geschaffen werden können, dies fällt jedoch aus dem Rahmen dieser Arbeit. Die Arbeit stellt dabei den Softwareentwicklungsprozess von der Projektidee über die Erhebung von Anforderungen und Entwicklung der zu Grunde liegenden Modelle bis zur Implementation dar.

Die Arbeit ist in die folgenden Abschnitte gegliedert:

Grundlagen
: Stellt die fachlichen Grundlagen zu dieser Arbeit aus den Bereichen den Softwareentwicklung sowie grundlegendes Wissen zu OER vor.

Problemanalyse
: Fasst den vorliegenden Sachverhalt bzw. die Problemstellung zusammen und ordnet die Grundlagen zu OER in den Projektkontext ein.

Systementwurf
: Arbeitet aufbauend auf der Problemstellung die Anforderungen an das Gesamtsystem heraus und schafft einen Überblick über das umzusetzende System.

Systemumsetzung
: Bildet die Entwicklungsphase des Projektes ab und geht dabei sowohl auf die verwendeten Werkzeuge und Komponenten als auch auf Teile des Softwaresystems gesondert ein.

Ausblick
: Schafft einen Ausblick auf mögliche Fortsetzungsprojekte und Anknüpfungspunkte für neue Arbeiten im Themenkomplex.

Fazit
: Zieht ein Fazit aus der Arbeit, insbesondere auch der Umsetzung des Projektes und schließt diese ab.

Kapitel 2

Grundlagen

Diese Arbeit baut inhaltlich und methodisch auf Konzepten aus der Softwaretechnik, vergleichbar mit den Inhalten aus den Grundlagenmodulen Softwareprojekt I + II, auf. Im Speziellen wird hier auf das Architekturmuster Model-View-Controller (MVC) (Abschnitt 2.2) eingegangen und die strukturellen Unterschiede zwischen klassischen Softwareprodukten und Webanwendungen herausgearbeitet. Darüber hinaus werden Open Educational Ressources (OER) als Kern der Anwendungsdomäne und Motivator dieser Arbeit behandelt (vgl. Abschnitt 2.1). Die zur Umsetzung des Projektes verwendeten Frameworks werden in Abschnitt 2.3 vorgestellt und grundlegend eingeführt.

2.1 Open Educational Ressources

OER bilden eine wichtige Grundlage für die Aufbereitung von Lehr- und Lerninhalten in der heutigen Zeit. Eine grundlegende Definition von OER wird im Verlauf der Arbeit durch die Rolle von Open Educational Ressources im Projektkontext ergänzt. Wichtig für die Betrachtung von OER ist, die möglichen Lizenzmodelle zu kennen und einschätzen zu können (vgl. Abschnitt 2.1.2 *Offene Lizenzmodelle - Creative Commons* und Abschnitt 2.1.3 *Lizenzierungsempfehlung*), um das passende Lizenzmodell für die selbstentwickelten Lehrmaterialien zu wählen.

2.1.1 Definition von OER

Die UNESCO definiert OER wie folgt:

» Open Educational Resources (OER) sind Bildungsmaterialien jeglicher Art und in jedem Medium, die unter einer offenen Lizenz stehen. Eine solche Lizenz ermöglicht den kostenlosen Zugang sowie die kostenlose Nutzung, Bearbeitung und Weiterverbreitung durch Andere ohne oder mit geringfügigen Einschränkungen. Dabei bestimmen die Urheber selbst, welche Nutzungsrechte sie einräumen und welche Rechte sie sich vorbehalten.

Open Educational Resources können einzelne Materialien, aber auch komplette Kurse oder Bücher umfassen. Jedes Medium kann verwendet werden. […] Kursmaterialien, […] Multimedia-Anwendungen […] – all diese Ressourcen sind OER, wenn sie unter einer offenen Lizenz veröffentlicht werden. « (UNESCO 2017)

Davon ausgehend fallen unter diese Definition bei entsprechender Lizensierung (siehe Abschnitt 2.1.2 *Offene Lizenzmodelle - Creative Commons*) Lehr- und Lerninhalte bzw. Material zur Erzeugung dieser, wie sie durch Nutzung des im Rahmen dieser Arbeit zu entwickelnden Systems gesammelt werden sollen. Des Weiteren kann das zu entwickelnde System (siehe Abschnitt 3.1 *Projektvorhaben*) inklusive der Rohdaten unter einer offenen Lizenz veröffentlicht werden. Dies ermöglicht eine weitere unabhändige Entwicklung auch über den Abschluss des Projektes hinaus.

2.1.2 Offene Lizenzmodelle - Creative Commons

Für Inhalte, die als Open Educational Ressources (OER) zur Verfügung gestellt werden sollen, werden die Lizenzmodelle nach Creative Commons (CC) empfohlen. Diese Standard-Lizenzverträge geben eine Reihe von Nutzungsmöglichkeiten und sind in ihrer Offenheit stark unterschiedlich. Die CC-Lizenzen bauen dabei auf den vier Modulen aus Tabelle 2.1 auf: *BY*, *NC*, *ND* und *SA*.

Icon[1]	Kürzel	Beschreibung
	BY	Die Urheber*in des Werkes und die Lizenz müssen in geeigneter Weise genannt werden.
	NC	Das Werk darf nur nichtkommerziell weiter verwendet werden.
	ND	Das Werk darf nicht verändert werden.
	SA	Eine Weiterverarbeitung des Werkes muss erneut unter der selben Lizenz veröffentlicht werden.

Tabelle 2.1 Die Module der CC-Lizenzen

Aus den Lizenzmodulen gemäß Tabelle 2.1 ergeben sich sieben Standardlizenzverträge. Hierbei kommt CC0 eine Sonderrolle zu, da damit alle Rechte (soweit möglich) abgetreten werden und Werke damit als gemeinfrei erklärt werden (sie sind dann in der public domain). Alle übrigen Lizenzen beinhalten immer mindestens die Namensnennung des Urhebers (*BY*) und können jeweils mit *SA*, *NC* und *ND* kombiniert werden, dabei schließen sich *ND* und *SA* (da hier eine Weiterverarbeitung angenommen wird, die durch *ND* nicht möglich ist) gegenseitig aus. Es ergeben sich somit die Lizenzmöglichkeiten aus Abbildung 2.1, welche nach ihrer Offenenheit mit den dadurch möglichen Nutzungsformen in der Grafik dargestellt werden.

[1] Icons bereitgestellt durch: Creative Commons, *https://creativecommons.org/about/downloads*, abgerufen am 12.04.2019, lizenziert unter Creative Commons BY

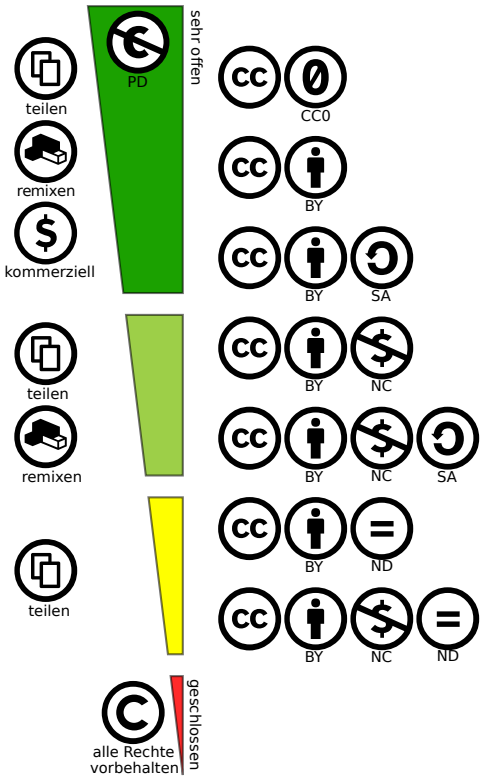

Abbildung 2.1 Die CC-Lizenzen angeordnet nach ihrer Offenheit (Muuß-Merholz 2017)

2.1.3 Lizenzierungsempfehlung

Unter der Vielzahl möglicher Lizenzen, zu denen neben den CC-Lizenzen auch die *MIT-Lizenz*[1], andere freie Lizenzmodelle sowie individuelle Lizenzverträge gehören, gibt es eine klare Tendenz zu den in Abschnitt 2.1.2 vorgestellten CC-Lizenzen (e-teaching.org 2018). Zur Verwendung in Bezug auf OER wird bspw. in Norwegen auf das Modell CC-BY oder CC-BY-SA gesetzt (Müller 2019a). Lizenzen mit NC-Bestandteil werden nicht berücksichtigt, da auch die Nutzung innerhalb von Schulen in freier Trägerschaft bereits als kommerzielle Nutzung ausgelegt werden kann (Müller 2016). Durch die mehrfache Übernahme, Zusammenstellung und Abwandlung von Materialien unter CC-BY Lizenzen steigt die Anzahl der anzugebenen Urheber*innen und somit der Aufwand für Lehrkräfte. Da die Bereitschaft der Lehrkräfte ihre so erzeugten Materialien ebenfalls als OER zur Verfügung zu stellen dadurch abnehmen dürfte wird eine Empfehlung für CC-0 ausgesprochen (Müller 2016). In der Tat wird »die Lizenzfrage auch als Barriere für das Teilen von Inhalten durch Lehrkräfte thematisiert« (Müller 2019a, S. 24). Hierbei können Urheber*innen optional dennoch genannt werden um so bspw. auch Anerkennung zu zeigen und einen Ansatz für ein Vertrauenskonzept in die Quallität zu geben. So ist die Veränderung bzw. Weiterverar-

[1] *https://de.wikipedia.org/wiki/MIT-Lizenz*, abgerufen am: 21.06.2019

beitung uneingeschränkt möglich und es können auf dieser Grundlage neue Lehrmaterialien entwickelt werden. Dies ermöglicht eine freie Verwendung der Materialien durch Lehrkräfte und Schüler*innen aller Schulformen und bspw. auch in Standardwerken von Schulbuchverlagen. Letzteres kann auch zu einer weiteren Verbreitung und zu einem höheren Bekanntheitsgrad von OER im Allgemeinen führen.

2.2 Softwareentwicklungmodell

Ein sehr verbreitetes Artchitekturmuster in Softwareentwicklungsprozessen ist Model-View-Controller (MVC). In der Entwicklung von Webanwendungen werden abgewandelte Versionen dieses Standardmusters verwendet um den veränderten Anforderungen und Möglichkeiten Rechnung zu tragen.

2.2.1 Das klassische MVC Muster

Im klassischen MVC Architekturmuster, wie in Buschmann et al. 1996 und Wikibooks 2018 beschrieben, werden die Komponenten in die drei Bereiche Model, View und Controller aufgeteilt. Hierbei kommt jeder Gruppe eine eigene Aufgabe zu, welche weitestgehend abgekapselt konstruiert ist, sodass einzelne Komponeneten leicht erweitert oder ausgetauscht werden können. Eine Kommunikation zwischen Model und View wird hierbei über das Beobachter-Entwurfsmuster abgebildet.

Die Gliederung der Aufgaben in die Bereiche Model, View und Controller stellt sich wie folgt dar:

Model
: Das Model dient der Repräsentation der Daten, diese enthalten keine weitergehende Geschäftslogik und werden lediglich von der View zur Darstellung verwendet. Alle Änderungen an den gespeicherten Daten werden durch den Controller durchgeführt. Es sind in objektorientierten Programmiersprachen auch Vererbungen innerhalb der im Model betrachteten Typen möglich.

View
: Die View stellt die Daten des Models dar und ermöglicht die Interaktion mit den Benutzer*innen. Eine Interaktion mit Nutzer*innen wird dem Controller mitgeteilt, sodass die Anzeige ebenfalls ohne Geschäftslogik auskommt. Die Anzeige ist dabei häufig eine Komposition aus mehreren Teilkomponenten. In der selben Anwendung kann es mehrere Anzeigen für verschiedene Aufgaben geben, bspw. zum Anzeigen der Daten in der Übersicht oder einer Detailsicht, sowie zum Bearbeiten.

Controller
: Der Controller nimmt Anweisungen durch Interaktion mit den Benutzer*innen von der View entgegen und verwaltet die Daten des Models. Sämtliche Geschäftslogik befindet sich daher im Controller, nur hier werden Manipulationen an den in der Anwendung gespeicherten Daten vorgenommen. Auch in den Klassen der Controller ist bspw. eine Vererbung möglich.

2.2.2 MVC im Web

In der Konstruktion von Webanwendungen nach dem MVC-Muster muss auf andere Aspekte eingegangen werden (Diepenmaat und 't Veer 2005). Teile der klassischen Umsetzung des Architekturmusters lassen sich auf Webanwendung nicht bzw. nur kaum anwenden, so wird in der Regel das Beobachter-Entwurfsmuster nicht verwendet um die View (die im Browser angezeigte Website) auf Änderungen des Models reagieren zu lassen, dies geschieht in der Regel durch Neuladen der entsprechenden Seite.

Insbesondere auf den Webbereich bezogen gibt es folgende Erweiterungen zum klassischen MVC (Diepenmaat und 't Veer 2005):

Model
: Dem Model sind die in der grafischen Benutzungsoberfläche eingesetzten Sprachen, wie HTML, JavaScript oder CSS unbekannt und haben für die Implementierung der zum Model gehörigen Klassen keine Bedeutung. Außerdem ist die Art und Form der Anfrage an den Webserver unerheblich für die Datenhaltung im Model.

View
: Im Webkontext werden für die View vor allem Templates (bezogen auf Ruby on Rails in z. B. Haml[2] oder ERB[3]), aus denen dann HTML generiert wird, verbreitet. Solche Templates können auch geschachtelt werden, was eine leichte Wiederverwendung und Erweiterung einzelner Komponenten ermöglicht.

Controller
: Für die Abhandlung der Anfragen an die Anwendung ist der Controller zuständig. Dieser wählt die passende View und die notwendigen Daten aus dem Model aus. Weiterhin nimmt der Controller Änderungen (Löschungen, Manipulationen) an den Daten vor. Der Controller kann dabei auch in mehrere Klassen aufgeteilt sein wie es in Ruby on Rails mit der Aufteilung auf grob eine Controller-Klasse je Datenklasse und der Zuordnung der jeweils zuständigen Controller-Implementierung durch die »routing«-Schnittstelle der Fall ist. Da der Controller alle Anfragen entgegen nimmt, wird in dieser Schicht auch beispielsweise die Authentifizierung und Autorisierung umgesetzt.

Ein möglicher Fallstrick bei der Implementation ist immer die Frage: *Wo ist dieser Code-Abschnitt unterzubringen?* Ist nicht klar zu entscheiden, zu welchem Bereich eine Funktion, Klasse, o. Ä. zuzuordnen ist können bspw. »Helper«-Klassen zum Einsatz kommen (Diepenmaat und 't Veer 2005).

[2] *http://haml.info/about.html*, abgerufen am: 01.04.2019
[3] *https://www.stuartellis.name/articles/erb/*, abgerufen am: 01.04.2019

2.3 Web-Frameworks

Zum Zweck der Umsetzung des Projektvorhabens dieser Bachelorarbeit wird ein Full-Stack Framework genutzt, um bei der Entwicklung Aufwand und damit Entwicklungszeit einzusparen. In verschiedensten Programmiersprachen sind solche Full-Stack Frameworks umgesetzt. Zu den bekanntesten solcher Frameworks auf unterschiedlichen Plattformen gehören z. B.: Symfony[4] oder Laravel[5] in PHP, Django[6] in Python, Rails[7] in Ruby, JavaEE[8] und Spring[9] in Java oder ASP.NET[10] in .NET.

Im Zuge dieser Arbeit wird Ruby on Rails als Framework verwendet und das Projekt entsprechend in Ruby umgesetzt. Die Wahl von Framework und Sprache ist vor allem aufgrund von persönlichen Erfahrungen gefallen, ein ausführlicher Vergleich der Frameworks findet im Rahmen dieser Arbeit nicht statt. Für die Entwicklung der grafischen Benutzungsoberfläche wird Bootstrap als Grundlage gewählt.

2.3.1 Ruby on Rails

Ruby on Rails (Rails) ist ein Web-Framework für die Programmiersprache Ruby und in dieser geschrieben. Entsprechend der für Ruby geläufigen Entwicklungsprinzipien wird im Zuge der Implementierung auf eine Vielzahl an Bibliotheken zurückgegriffen. Das Framework setzt auf das verbreitete MVC Architekturmuster und fördert dessen Einhaltung durch vorgegebene Strukturen. Diese Strukturierung und allgemein die Programmiersprache Ruby führen zu relativ kompakten Quellcode-Abschnitten für die einzelnen Module.

Im Kontext von Rails kommen die Komponenten ActiveRecord im Model, ActionView in der View sowie ActionController und ActionDispatch gemeinsam im Controller zum Einsatz. ActionView, ActionController und ActionDispatch werden dabei auch gemeinsam als ActionPack bezeichnet (Morsy und Otto 2012). Bei der Verwendung von ActiveRecord ist dabei die Datenquelle, d. h. ob die Daten aus einer entfernten Web-API, einer relationalen Datenbank o. Ä. stammen, für die Implementierung unerheblich. Es dient als Verknüpfung zwischen den objektorientierten Klassen in der Anwendung und beispielsweise den Tabellen in einer relationalen Datenbank. Die Komponente ActionDispatch ist für die Zuordnung an die zuständigen Controller-Klassen und ihre »Actions« (Methoden) verantwortlich (das so genannte »routing«, vgl. Abschnitt 2.2.2 *MVC im Web* zu *Controller*). Der ActionController fungiert als Oberklasse für alle Controller-Klassen und damit ist diese Komponente für Manipulationen am Model, Caching, Session-Management usw. zuständig. Die Erstellung von Templates wird in der Komponente ActionView umgesetzt, dabei sind Ausgaben in verschiedenen Formaten wie HTML oder JSON möglich (Morsy und Otto 2012).

[4] *https://symfony.com/what-is-symfony*, abgerufen am: 01.04.2019
[5] *https://laravel.com/*, abgerufen am: 01.04.2019
[6] *https://www.djangoproject.com/start/overview/*, abgerufen am: 01.04.2019
[7] *https://rubyonrails.org/everything-you-need/*, abgerufen am: 01.04.2019
[8] *https://javaee.github.io/*, abgerufen am: 09.07.2019
[9] *https://spring.io/*, abgerufen am: 01.04.2019
[10] *https://dotnet.microsoft.com/apps/aspnet*, abgerufen am: 01.04.2019

Rails verfolgt das Paradigma »Konvention statt Konfiguration« und setzt dabei auf sinnvolle Standards für bspw. Dateipfade und Konfigurationsmöglichkeiten. Als Beispiel seien für die Persistenz die Standardwerte der Tabellennamen (abgeleitet aus den Klassen des Models) genannt sowie die Konvention, dass jede Tabelle einen Primärschlüssel *id* besitzt. Außerdem erstrecken sich die Namenskonventionen auch auf das Dateisystem, so geben die Controller-Klassenbezeichnungen die Orderstruktur für die View-Komponenten vor und die »Actions« der jeweiligen Klassen die zugehörigen Dateien im beschriebenen Verzeichnis. Diese Konventionen ermöglichen es, Anwendungen sehr kompakt zu entwickeln. Dies sorgt außerdem dafür, dass sich Rails-Entwickler schnell in neue unbekannte Projekte einfinden können. Natürlich bleibt es bei allen Konventionen weiterhin möglich, davon abweichende Konfigurationen vorzunehmen um die Entwicklung nach den eigenen Vorstellungen anzupassen (Morsy und Otto 2012).

2.3.2 Bootstrap

Bootstrap wurde ursprünglich von zwei Entwicklern bei Twitter entworfen und als internes Projekt »Twitter Blueprint« gepflegt. Dieses Projekt wurde schnell weiterentwickelt und hat in weiteren Versionen z. B. eine native Unterstützung für responsive Layouts nach dem Ansatz »mobile first« erhalten (Bootstrap-Team 2019a). Für die Gestaltung responsiver Layouts arbeitet Bootstrap dabei mit »breakpoints« an mehreren definierten Bildschirmbreiten[11].

Grundlage der Layouts in einer Anwendung bzw. Homepage mit Bootstrap ist das Container-Modell mit einem Rastersystem. Das Rastersystem besteht aus Zeilen (`rows`) und darin enthaltenen Spalten (`cols`). Als Beispiel für das Verhalten dieser Rasterelemente wird nachfolgend das »Column wrapping« betrachtet: Dabei kann eine Zeile bis zu 12 Spalten auf die gesamte Breite berechnet enthalten. Die Breite der Spalten kann über definierte CSS-Klassen (i. d. R. gleichverteilt im Elternelement) festgelegt werden. Werden zu viele Spalten in einer Zeile gesetzt (bzw. insgesamt die Summe der Breiten zu groß) werden Spalten, welche die Breite der Zeile überschreiten, in die nächste Zeile verschoben, dort agieren die Spalten wie gewohnt weiter. Dieses Verhalten kann in Abbildung 2.2 und dem dazugehörigen Quellcode 2.1 betrachtet werden (Bootstrap-Team 2019b).

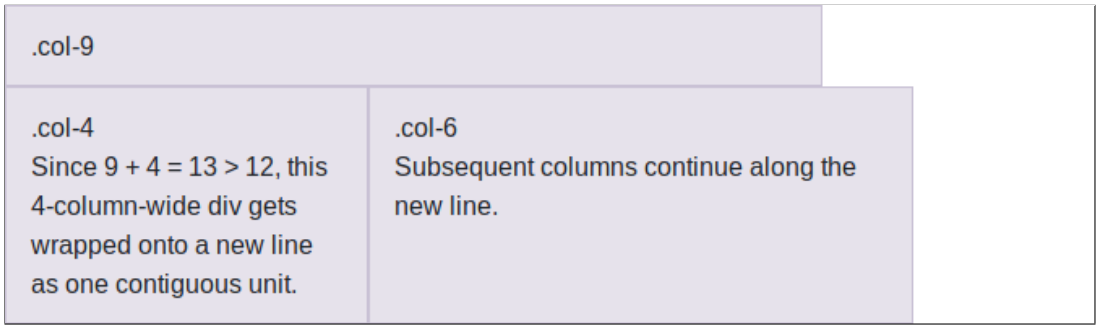

Abbildung 2.2 Column wrapping in Bootstrap (Bootstrap-Team 2019b)

[11] *https://getbootstrap.com/docs/4.3/layout/overview/#responsive-breakpoints*, abgerufen am: 20.06.2019

```
<div class="container">
  <div class="row">
    <div class="col-9">.col-9</div>
    <div class="col-4">.col-4<br>Since 9 + 4 = 13 &gt; 12, this 4-column-wide ↩
        div gets wrapped onto a new line as one contiguous unit.</div>
    <div class="col-6">.col-6<br>Subsequent columns continue along the new line↩
        .</div>
  </div>
</div>
```

Quellcode 2.1 Column wrapping in Bootstrap (Bootstrap-Team 2019b)

Weitere Details der Oberflächengestaltung unter Bootstrap können der offiziellen Dokumentation[12] entnommen werden. Für die Umsetzung des Systems (vgl. Abschnitt 5.2) werden noch einige weitere Funktionen und Gestaltungselemente von Bootstrap verwendet.

[12] *https://getbootstrap.com/docs/4.3/*, abgerufen am: 20.06.2019

Kapitel 3

Problemanalyse

Zur Analyse des einem Softwareprojekt zu Grunde liegenden Problems wird das Projektvorhaben (oder auch Projekt- oder Produktidee) betrachtet und in Relation zum Ist-Zustand gesetzt. Im Rahmen dieser Arbeit wird auch die besondere Rolle von Open Educational Ressources im Projektkontext noch einmal beleuchtet.

An die Problemanalyse gliedert sich direkt der Systementwurf an, in welchem aus dem Projektvorhaben die konkreten Anforderungen herausgearbeitet werden und eine Grundlage für die Implementation geschaffen wird.

3.1 Projektvorhaben

Die Anforderungen gehen aus dem Projekt »#UnsereWörter« von Prof. Dr. Frank J. Müller am Fachbereich 12 der Universität Bremen hervor bzw. sind mit ihm erarbeitet worden. Im Sinne dieser Arbeit als Projekt der Softwareentwicklung ist Prof. Müller als Kunde/Stakeholder zu sehen.

Das System soll Daten zu Wörtern der deutschen Sprache, neben Wörtern aus Grund- und Orientierungswortschätzen der Bundesländer auch thematisch erhobene Wortschätze von Kindern und Jugendlichen aufnehmen. Dabei werden diese Daten untereinander verknüpft und mit Metadaten neben der reinen Schreibweise angereichert. Diese Metadaten können bspw. Verknüpfungen zu Stichworten oder Einordnungen in Kategorien sein, sodass bspw. eine Suche nach »einem großen, weißen Tier« unter Anderem das Wort »Eisbär« als Ergebnis liefert oder eine Filterung anhand der vorkommenden Buchstaben (häufiger Anwendungsfall im ersten Schuljahr) möglich wird. Eine erste Übersicht der angeforderten Datenstrukturen ist im Anhang unter A.9.1 beigefügt.

Die Daten sind dabei durch Studierende, Lehrer*innen und gegebenefalls in Zukunft Schüler*innen eintragbar und natürlich abrufbar. Dabei sollen sämtliche Daten und Metadaten zunächst nur für vorab autorisierte Nutzer*innen editierbar sein um die Datenbank initial zu befüllen. Im späteren Projektverlauf soll diese Möglichkeit öffentlich zugänglich sein und die Möglichkeit gegeben werden aus diesen

Datensätzen Unterrichtsmaterialien als OER umzusetzen und erneut bereit zu stellen. Diese Abruf- und Editiermöglichkeiten sollen auch durch externe Anwendungen wie z. B. Apps eingebunden werden können, hierfür muss entsprechend eine Programmierschnittstelle bereit gestellt werden, welche alle Funktionalitäten der Weboberfläche abbildet.

3.2 Ist-Zustand

Zum Zeitpunkt des Projektauftaktes existiert keine zentrale Software-Lösung zur Erfassung und Strukturierung der Daten mit dem Umfang und Möglichkeiten welche der Projektidee zugrunde liegen. Große kommerzielle Datenbanken wie beim Duden[1] existieren, sind jedoch mindestens für die professionelle Nutzung mit teilweise sehr hohen Kosten verbunden. Offene Datenbanken wie z. B. Wiktionary[2] bieten zum Teil unpassende Lizenzbedingungen zur Weiterverwendung der Daten oder sind nicht maschinenverarbeitbar und scheiden somit einer weiteren Verwendung und Verknüpfung aus. Die Plattform ist nicht in Konkurrenz zu bestehenden OER-Verzeichnissen und -Services[3] zu sehen, sondern als weitere Ergänzung dieser Angebote um Grundlagenmaterialien.

Die zu erfassenden Daten liegen zum Teil als Tabellen (CSV) vor bzw. sind nicht aggregiert und können in dieser Form nicht ausreichend weiter verarbeitet werden. Außerdem liegen zu vielen Wörtern nur die Grundformen vor und bspw. Konjugationen und weitere Metadaten fehlen vollständig. Inbesondere ist auch eine Verknüpfung der Datensätze untereinander (z. B. Stichwörter, Reimwörter, usw.) in der aktuellen Form nicht möglich. An der aktuellen Datensammlung kann des Weiteren nicht bzw. nur schwer kollaborativ gearbeitet werden. Aus den Anforderungen ist eine Übersicht der benötigten Datenstrukturen erarbeitet worden (vgl. Abschnitt A.9.1 *Datenbankstruktur, Initialversion*) welche in Abschnitt 4.4 *Datenmodell* aufgegriffen wird.

[1] *https://www.duden.de*, abgerufen am: 21.06.2019
[2] *https://de.wiktionary.org/wiki/Hilfe:Hinweise_f%C3%BCr_Leser*, abgerufen am: 21.06.2019
[3] *https://open-educational-resources.de/materialien/oer-verzeichnisse-und-services/*, abgerufen am: 21.06.2019

3.3 Open Educational Ressources im Projektkontext

Die Inhalte, welche im Rahmen der zu entwickelnden Umgebung erfasst werden sollen sind gemäß Abschnitt 2.1 als OER zu betrachten. Im Rahmen dieser Projektarbeit ergeben sich weitere Eigenschaften und Vorteile im Einsatz von OER, auf welche im Folgenden detailierter eingegangen wird.

3.3.1 Vorteile von Open Educational Ressources

Die Plattform und deren Inhalte als OER zu veröffentlichen bzw. die Sammlung der (Meta-)Daten mittels einer Webplattform bringt auf mehreren Ebenen Vorteile:

Studierende nutzen digitale Medien bereits stark für private und studienbezogene Aktivitäten, daher ist davon auszugehen, dass eine solche Plattform auch bei herangehenden Lehrkräften im Studium eine hohe Akzeptanz erfahren wird und so sowohl für eine Phase der Datenerfassung als auch für die tatsächliche Nutzung im Lehrkontext als Vorbereitungs- bzw. Unterstützungsanwendung von den Studierenden verwendet wird. Hierbei sind der Unterhaltungsfaktor sowie die Usability wichtige Merkmale um den Erfolg bzw. die Akzeptanz eines solche Systems zu gewährleisten. Zusätzlich spielt es eine Rolle, wie die Studierenden bereits in ihrem Studium an ein System herangeführt und mit diesem sozialisiert werden, daher ist eine Nutzung bereits im Studium z. B. im Rahmen von Seminaren gut vorstellbar und sinnvoll. (Steffens et al. 2018)

Aus der Perspektive von Schüler*innen kann man feststellen, dass durch die Nutzung von digitalen Medien in Freizeit sowie in der Schule dazu führt, bzw. die Anforderung hat, dass ein Umgang damit auf einer lockeren Ebene geschieht. Hierdurch kann das klassische formelle Lernen durch eine informelle Form ersetzt bzw. ergänzt werden. So ist eigenständiges und eigenverantworliches Lernen eine Notwendigkeit und kann durch den gezielten Einsatz von Digitalen Medien gefördert werden. So ist der Einsatz von einer eigentlich zum Lernen gedachten Plattform auch im Privatumfeld denkbar, wenn diese einen Anreiz zur Auseinandersetzung mit dem Thema geben. Ein solcher Anreiz könnte bspw. durch Gamification erlangt werden.

3.3.2 Einsatz von Open Educational Ressources in der Schule

Bereits heute ist die Erstellung von Unterrichtsmaterialien durch Lehrkräfte eine Kernkompetenz und -aufgabe im Arbeitsalltag. Viele aufbereitete Materialien werden dabei nur für den eigenen Unterricht erarbeitet und finden darüber hinaus insbesondere außerhalb der konkreten Schule keine Anwendung. Eine Umsetzung dieser Materialen als offene Lehr- und Lerninhalte – also OER – ist als nächster Schritt anzusehen. Das für die Erstellung von OER-Inhalten auf ebensolche als Grundlage zurückgegriffen wird ist nur ein logischer Schluss, daher sollten auch Grundlagenmaterialien, angereicherte und aggregierte Daten zur Verwendung in diesem Kontext frei verfügbar und einfach verwendbar sein. Für den Einsatz von OER im Schulkontext müssen natürlich die technischen Grundlagen geschaffen werden. So ist bspw.

der Aufbau von offenen Plattformen zur Erarbeitung und Veröffentlichung von OER-Materialien ein wichtiger und notwendiger Schritt. Ein Baustein in diesem Kontext bildet die mit diesem Projekt zu erarbeitende Plattform. (Brause und Schulz 2017)

Ein wichtiger Punkt, der durch den Einsatz von OER-Materialien im Unterricht erreicht werden kann, ist der Ausbau der Medienkompetenz der Schüler*innen – insbesondere auch auf Urheberrechts und Lizenzfragen. Der Ausbau der Fähigkeiten und Erfahrungen der Schüler*innen in diesem Bereich ermöglicht es ihnen dieses Wissen für eigene Ausarbeitungen von bspw. Hausarbeiten und Präsentationen einzubringen. Gleichzeitig ermöglicht dies den Schüler*innen und Lehrkräften eine steigende Rechtssicherheit im Umgang mit den Quellen. (Kück 2017)

Schüler*innen sollen im Unterricht auch die Möglichkeit erhalten, eigene Inhalte aufgabenorientiert und unter der Maßgabe sinnvoller Quellenarbeit zu erarbeiten. Dies kann z. B. durch die Weiterverarbeitung von OER-Materialien erreicht werden, da hierbei die lizenzrechtlichen Aspekte für die Schüler*innen transparent und einfach zu beachten sind. (Kück 2017) Bezogen auf das konkret im Rahmen dieser Arbeit umzusetzende Projekt ließen sich eigene Inhalte, wie beispielsweise thematische Wortsammlungen, Zuordnungen von Stichworten oder individuelle Kategorisierungen aus den bereits vorhandenen (Meta-) Daten erarbeiten. Diese so erzeugten Inhalte bzw. mit weiteren Metadaten angereicherten Materialien ließen sich wiederum als OER veröffentlichen und in die vorhandene Datenbank eingliedern.

Kapitel 4

Systementwurf

Im Systementwurf werden die Anforderungen aufgearbeitet und analysiert, hierdurch kann eine Definition typischer Anwendungsfälle erfolgen, womit abschleißend die Systemarchitektur und das Datenmodell erarbeitet wird.

4.1 Anforderungen

Ergänzend zu den Anforderungen aus der Problemanalyse (siehe Abschnitt 3.1 *Projektvorhaben*) werden in diesem Abschnitt zusätzliche, zum Teil technische, Anforderungen formuliert.

Das System soll als Webanwendung umgesetzt werden, damit diese ohne eigenen Installationsaufwand von möglichst vielen potenziellen Nutzer*innen auf unterschiedlichen Systemen direkt genutzt werden kann. Das System wird vorerst mit einer zentralen Instanz geplant, Möglichkeiten zur Lastverteilung und Skalierbarkeit werden auf Ebene des Webservers bzw. der Datenbank ausgelagert und im Rahmen dieser Projektarbeit nicht einbezogen. Dieser Ansatz einer zentralisierten Webanwendung erleichtert zudem ansonsten aufkommende Fragestellungen zur Aktuallität der Daten bzw. Synchronisation zwischen den Instanzen.

Da die Rohdaten dieses Systems auch an anderen Stellen z. B. zum Erzeugen von Unterrichtsmaterialien oder zur Darstellung in einer interaktiven App für die Schüler*innen verfügbar bzw. verarbeitbar sein sollen, muss eine entsprechende Schnittstelle hierfür bereit gestellt werden. Diese Programmierschnittstelle soll hierbei sowohl lesend auch auch schreibend verfügbar gemacht werden (vgl. Abschnitt 4.1.2 *Systemzugang und Berechtigungen*). Die Berechtigungen sind dabei denen des allgemeinen Systemzugriffs gleichgesetzt.

4.1.1 Datenerfassung

Die zu erfassenden Daten umfassen Wörter mitsamt allen Metadaten und Verknüpfungen untereinander. Hierbei muss der Import bestehender und noch zu erarbeitenden Ausgangsdaten möglich sein und außerdem ein weiteres Bearbeiten und Verknüpfen durch die Nutzer*innen auf eine einfache Art durchführbar sein. Die Metadaten umfassen hierbei klassisch zu Wörtern zuordenbare Daten wie Konjugationen und Pluralbildungen, aber auch Beispielsätze, Bilder und Informationen über möglicherweise aufkommende Lernschwierigkeiten von Schüler*innen im Umgang mit diesen Wörtern.

Eine initial erarbeitete Liste mit groben Anforderungen an die zu verarbeitenden Daten kann der Anlage in Abschnitt A.9.1 entnommen werden. Für die Erarbeitung des Datenmodells (vgl. Abschnitt 4.4) soll im Rahmen der Bachelorarbeit nur ein Auszug dieser Datenfelder verwendet werden, um die initiale Projektkomplexität möglichst gering zu halten und somit das System in Teilgebieten bereits schnell produktiv nutzbar zu machen.

4.1.2 Systemzugang und Berechtigungen

Die Anforderungen an den Zugriff auf das System und die in diesem gehaltenen Daten ist abhängig vom Einsatzszenario bzw. dem Stand der Projektphase. Da das System bereits während der Entwicklungsphase mit einem Teil der geforderten Funktionen und Datenfelder eingesetzt werden soll, ergibt sich hier ein anderes Szenario als zum geplanten produktiven Einsatz nach Abschluss aller (absehbaren) Entwicklungsaufgaben. Während dieser Entwicklungszeit soll das System ausschließlich für registrierte Benutzer*innen zugänglich sein. Dabei muss sowohl der lesende als auch der schreibende Zugriff erst hinter einer Loginmaske verfügbar sein. Es sollen vorerst drei Berechtigungsstufen für die Benutzer*innen vorgesehen werden: »Lesende Benutzer*innen« mit ausschließlich lesendem Zugriff auf alle hinterlegten Datensätze und der verknüpften Metadaten. »Normale Benutzer*innen« mit Zugriff auf alle Datensätze zu den hinterlegten Wörtern inklusive der verknüpften Metadaten und die Bearbeitung dieser. Außerdem soll es den Benutzer*innen möglich sein, neue Datensätze anzulegen. »Administrative Benutzer*innen« erhalten darüber hinaus die Möglichkeit, Massenoperationen auf den Datensätzen durchzuführen (bspw. der Zuweisung von Stichwörtern). Der Import von Wortschatzlisten aus externen Quellen soll ebenfalls ausschließlich dieser Benutzer*innengruppe möglich sein. Daneben sollen Administrator*innen die Möglichkeit erhalten, die Benutzer*innen des Systems zu verwalten, allgemeine Einstellungen vorzunehmen und Datensätze endgültig zu löschen. Die Accountregistrierung bzw.-freischaltung soll nur durch Administrator*innen des Systems durch eine dafür zu schaffende Schnittstelle erfolgen.

Benutzer*innen sollen entweder lokal mit einer hinterlegten E-Mailadresse und einem Passwort angelegt werden können oder sich über den Single-Sign-On (SSO)-Dienst der Universität Bremen »Shibboleth«[1] authentifizieren. Voraussetzung für den Zugriff über Shibboleth soll sein, dass der Universitätsaccount vorab von einer Administrator*in des Systems freigeschaltet wird. Dies soll ebenfalls über die Accountver-

[1] *https://onlinetools.zfn.uni-bremen.de/server/content/aai/shibboleth-anwender.php*, abgerufen am: 16.04.2019

waltung umgesetzt werden. Um den Einsatz in von Prof. Dr. Frank J. Müller durchgeführten Lehrveranstaltungen zu erleichtern wird ebenfalls eine Funktion zum Massenfreischalten von Universitätsaccounts vorgsehen. Hierfür sollen aus Stud.IP[2] exportierte Teilnehmer*innenlisten als Datenbasis verwendet werden.

Nach Beendigung der Entwicklungsphase soll der lesende Zugriff im Gedanken von OER öffentlich zugänglich gemacht werden. Das Editieren der Datensätze wird dann ggf. über einen Review-Prozess implementiert. Dieser Schritt wird im Rahmen dieser Arbeit jedoch nicht bearbeitet.

Die zu schaffende Programmierschnittstelle (Application Programming Interface (API)) soll den Zugriff äquivalent zu den bereits beschriebenen Berechtigungen ermöglichen. Anstelle von Zugangsdaten mit E-Mailadresse und Passwort bzw. als Shibboleth-Login soll hierbei auf von jeder Benutzer*in (neu-)erzeugbare API-Tokens zurückgegriffen werden. Dies ermöglicht es jeder Benutzer*in den Zugriff per API (z. B. in eigenen Apps o. Ä.) umzusetzen und dabei gleichzeitig die Zuordnung der Aktionen auf den Account beizubehalten, um bspw. das in Abschnitt 4.1.4 beschriebene Punktesystem auch auf dieser Zugriffsebene zu nutzen. Die API-Tokens sollen hierbei über genau die Berechtigungen verfügen, welche die Benutzer*in auch über die grafischen Benutzungsoberfläche ausüben könnte. Dies bedeutet insbesondere, dass nur Administrator*innen-Tokens Löschoperationen durchführen dürfen und Tokens von lesenden Benutzer*innen ebenfalls nur einen lesenden Zugriff erhalten.

4.1.3 Erreichbarkeit

Das System soll von mehreren Nutzer*innen parallel genutzt werden können und auf einer gemeinsamen Datenbasis arbeiten. Der Zugriff muss von unterschiedlichen Systemen bezogen auf Betriebssysteme und Bildschirmauflösungen problemlos möglich sein. Daher ist das Projekt als Webanwendung mit einer zentralen Instanz umzusetzen. Diese Webanwendung soll dauerhaft verfügbar sein und über keine bzw. nur kleine Ausfallzeiten verfügen. Eine Lastverteilung bzw. Parallelisierung wird in der Implementierung des Systems nicht zur Umsetzung vorgesehen. Diese kann über den jeweiligen Web- oder Datenbankserver umgesetzt werden und wird im Rahmen der Bachelorarbeit nicht betrachtet. Eine Datensicherung soll ebenfalls direkt auf Ebene des Datenbanksystems umgesetzt werden. Dies kann bspw. durch Implementation eines Cronjob zum automatisierten Erzeugen einer Datenbanksicherung umgesetzt werden. Auf eine konkrete Umsetzung wird in dieser Bachelorarbeit nicht eingegangen.

[2] *https://elearning.uni-bremen.de/*, abgerufen am: 16.04.2019

4.1.4 Gamification

Gerade in der Anfangsphase des Projektes wird eine der Hauptaufgaben des Systems darin bestehen, die enthaltende Datenbasis zu vervollständigen bzw. zu erweitern. Hierfür müssen die Nutzer*innen des Systems dazu animiert werden, möglichst viele (Meta-)Daten zum Datenschatz beizusteuern. Ein möglicher Anreiz hierzu kann über den Ansatz der Gamification geschaffen werden. Hierfür soll ein Punktesystem umgesetzt werden, bei welchem die Nutzer*innen für jede Änderung an einem Wort (hierbei zählt jedes veränderte Datenfeld oder jede hinzugefügte Verknüpfung als eine Änderung) einen Punkt auf ein internes Punktekonto gutgeschrieben bekommt.

Zusätzlich sollen Teams gebildet werden, welche sich didaktisch z. B. aus bereits gebildeten Gruppen in Seminaren ergeben. Dies bietet sich an, da Prof. Dr. Frank J. Müller das Systems in von ihm durchgeführten Seminaren an der Universität Bremen intensiv einsetzen möchte. Teams sollen hierbei ebenfalls eine gemeinsame Punktzahl angezeigt bekommen, welche sich aus der Summe der Punktzahlen aller Mitglieder*innen ergibt. Eine Bestenliste der Teams oder Einzelnutzer*innen wäre eine Möglichkeit, um sich selbst mit anderen Nutzer*innen zu vergleichen. Ebenfalls lassen sich über die Punktzahlen (als Indikator der Aktivität auf der Plattform) auch Kriterien für Studienleistungen in einem Seminar abbilden.

Dass sich ein solches Punktesystem durch wiederholte Änderungen aushebeln bzw. manipulieren lässt wird in Abstimmung mit Prof. Dr. Frank J. Müller als offenes Problem akzeptiert. Ein möglicher Ansatz zur Einschränkung von Manipulationen wäre bspw. für neue Felder eine höhere Punktzahl als für das Bearbeiten eines bereits ausgefüllten Feldes zu geben. Außerdem könnte das Entfernen eines Wertes zu einem Punkteverlust führen. Beide Maßnahmen können jedoch ebenfalls zur Folge haben, dass die Korrektur fehlerhafter Daten weniger attraktiv erscheint, wobei im Allgemeinen jedoch fehlerhafte Daten gravierender als fehlende Daten anzusehen sind.

Um das Ausfüllen von leeren Feldern zu erleichtern sollen Sortier- und Filterfunktionen geschaffen werden, welche die Interaktion mit Datensätzen, an denen »sich die Arbeit lohnt«, erleichtern und motivieren. Außerdem sollen in einer tabellarischen Übersicht leere bzw. unvollständige Felder prominent hervorgehoben werden.

4.1.5 Benutzungsoberfläche/UX

Die Oberfläche des Systems muss sowohl an einem großen Monitor als auch auf mobilen Geräten einfach zugänglich sein. Insbesondere ist auch die mobile Nutzung wichtig, um die Motivation unter Nutzer*innen zu halten, in der Freizeit (z. B. bei Bahnfahrten) an der Vervollständigung der Datenbasis zu arbeiten.

Es soll eine einfache, übersichtliche Oberfläche geschaffen werden, welche durch klare Strukturen die Arbeitsabläufe der Nutzer*innen erleichtert. Dabei müssen die Menüs und Eingabemasken über das gesamte System konsistent sein und auch in bspw. der Wortwahl nicht abweichen. Eine solche einheitliche Wortwahl lässt sich mittels Techniken zur Internationalisierung von Anwendungen leicht umsetzen. Hierbei werden alle Texte einer Anwendung als Platzhalter umgesetzt, welche durch die passende Übersetzung

in der angefragten Sprache ersetzt werden. Dieses Vorgehen ermöglicht zum einen mehrsprachige Anwendungen und sorgt zum Anderen für eine einheitliche Wortwahl, da sich an festgelegten »Übersetzungen« orientiert wird. Die Mehrsprachigkeit des Systems und der enthaltenen Datensätze ist ein langfristiges Ziel von Prof. Dr. Frank J. Müller. Da dies im Kontext dieser Bachelorarbeit jedoch noch keine Relevanz hat wird auf diesen Aspekt nicht weiter eingegangen (vgl. Kapitel 6 *Ausblick*). Sollte es in der Bedienung des Systems zu Fehlermeldungen kommen, müssen diese für jede Nutzer*in verständlich sein und eine zügige Weiterarbeit der Nutzer*in ermöglichen.

4.1.6 Erweiterbarkeit

Eines der Hauptziele ist der schnelle Start der Plattform. Dies soll wie in Abschnitt 4.1.1 beschrieben erreicht werden, indem initial lediglich ein kleiner Teil der geforderten Datenfelder implementiert wird. Durch dieses Vorgehen ist es zwingend notwendig, dass nachträglich Datenfelder, Verknüpfungsmöglichkeiten und Funktionen hinzugefügt werden können ohne Komplikationen zu bestehenden Funktionalitäten hervorzurufen. Dieses Vorgehen wird durch die initiale Implementierung der wichtigsten und gleichzeitig »einfachen« Datenfeldern gestaltet werden. Diese sollen direkt von den Nutzer*innen (in der Startphase werden dies vor allem Studierende sein) befüllt werden können. Komplexere Felder und Verknüpfungen, die teilweise zum Einpflegen auch Abhängigkeiten von weiteren externen Personen haben, werden im Laufe der Projektentwicklung hinzukommen. Im Rahmen dieser Bachelorarbeit ist die vollständige Fertigstellung des Projektes durch externe Abhängigkeiten nicht absehbar. Die bereitgestellte API muss hierbei mit der grafischen Benutzungsoberfläche mitwachsen und dieselben Funktionen bereitstellen. Dabei muss sich die API ebenfalls abwärtskompatibel verhalten. Um dies formalisiert umzusetzen, bzw. zu dokumentieren, wird ab der ersten Veröffentlichungsfassung, welche im Anschluss an die Bachelorarbeit erfolgen wird, eine Versionierung nach »SemVer«[3] vorgesehen.

[3] *https://semver.org/*, abgerufen am: 16.04.2019

4.2 Anwendungsfälle

Im Rahmen des Entwicklungsprozesses werden Anwendungsfälle definiert, welche eine formalisierte Darstellung typischer Einsatzszenarien darstellen. Solche Anwendungsfälle können bspw. auch als Grundlage für (automatisierte) Tests verwendet werden. Im Rahmen dieser Arbeit werden auf den folgenden Seiten 6 exemplarische Anwendungsfälle skizziert. Eine Aufzählung weiterer Anwendungsfälle würde über den Rahmen dieser Bachelorarbeit hinaus gehen. In der Darstellung wird ein tabellarisches Layout gewählt, welches die wichtigsten Eckpunkte des Anwendungsfalls aufführt.

Übersicht der Anwendungsfälle:

Anwendungsfall	**Datenabfrage mit Suche**
Akteure	Lehrkräfte, Datenpfleger*innen, Administrator*innen
Vorbedingung	• Benutzer*in angemeldet. • (vollständige) Datensätze sind in der Datenbank vorhanden.
Nachbedingung Erfolgsfall	Eine Liste zur Abfrage passender Datensätze wurde ausgeliefert.
Ausnahmen / Fehlerfälle	Keine passenden Datensätze vorhanden.
Nachbedingung Fehlerfall	Meldung, dass keine übereinstimmenden Datensätze gefunden wurden.
Ablauf	1. Die gewünschten Kriterien (bspw. ein Suchbegriff, eine Wortverknüpfung, eine Eigenschaft) werden in ein Formular eingegeben. 2. Das Formular wird abgesendet. 3. Die Ergebnisse werden ausgeliefert.
Varianten / Erweiterungen	• Unterschiedliche Wortarten implizieren unterschiedliche Filtermöglichkeiten, da sie über andere Datenfelder verfügen. • Ein Zugriff per API übersendet die Abfragedaten, das Formular aus Schritt 1 geht aus der Quellanwendung hervor.
Anmerkungen	

Tabelle 4.1 Anwendungsfall: Datenabfrage mit Suche

Anwendungsfall	**Import einer Datensammlung als CSV**
Akteure	Administrator*innen
Vorbedingung	• Benutzer*in mit administrativen Rechten angemeldet. • Die CSV enthält zeilenweise Datensätze im Format `Wortart,Grundform,Kurzbedeutung,Themen`.
Nachbedingung Erfolgsfall	Die in der CSV enthaltenen Datensätze wurden importiert.
Ausnahmen / Fehlerfälle	Die CSV ist nicht passend formatiert.
Nachbedingung Fehlerfall	Meldung, dass die Formatierung der CSV nicht dem geforderten Format entspricht.
Ablauf	1. Es wird eine CSV-Datei ausgewählt. 2. Metadaten zur Quelle (bspw. Titel, Autor*in, Lizenz) werden angegeben. 3. Das Formular wird abgesendet. 4. Die Einträge der CSV werden in der Datenbank angelegt und mit dieser Quelle verknüpft.
Varianten / Erweiterungen	• Einträge sind bereits in der Datenbank erfasst (identische Wortart, Grundform und Kurzbedeutung) → vorhandene Einträge werden um die importierten Themenzuordnungen erweitert und zusätzlich mit dieser neuen Quelle verknüpft.
Anmerkungen	Es sind auch weitere Quelldatenformate denkbar, das hier angegebene entstammt einer konkreten Idee bzw. Anforderung aus einem Gespräch mit Prof. Dr. Frank J. Müller.

Tabelle 4.2 Anwendungsfall: Import einer Datensammlung als CSV

Anwendungsfall	**Bearbeiten von Datensätzen**
Akteure	Datenpfleger*innen, Administrator*innen
Vorbedingung	• Benutzer*in angemeldet. • Datensätze in der Datenbank vorhanden.
Nachbedingung Erfolgsfall	Der gewählte Datensatz wurde wie gewünscht bearbeitet.
Ausnahmen / Fehlerfälle	Eine zweite Benutzer*in möchte den Datensatz ebenfalls bearbeiten.
Nachbedingung Fehlerfall	Die zuletzt ausgeführte Speicherung wird in die Datenbank übernommen (siehe Anmerkungen).
Ablauf	1. Der zu bearbeitende Datensatz wird aus der Übersicht (eingrenzbar gemäß Tabelle 4.1 *Anwendungsfall: Datenabfrage mit Suche*) zum Bearbeiten ausgewählt. 2. Die gespeicherten Daten werden in einem Formular dargestellt. 3. Die Benutzer*in nimmt die gewünschten Änderungen vor. 4. Das Formular wird abgesendet. 5. Die Daten werden aktuallisiert. 6. Die Benutzer*in erhält Punkte für die getätigten Änderungen gutgeschrieben und wird entsprechend informiert (siehe Abschnitt 4.1.4 *Gamification*).
Varianten / Erweiterungen	• Der zu bearbeitende Datensatz existiert noch gar nicht → siehe Tabelle 4.4 *Anwendungsfall: Datensatz anlegen*. • Es sollen mehrere Datensätze mit der selben Änderung bearbeitet werden → siehe Tabelle 4.5 *Anwendungsfall: Masseneditieren mehrer Datensätze*.
Anmerkungen	Das gleichzeitige Bearbeiten des selben Datensatzes von unabhängigen Benutzer*innen wird als wenig kritisch angesehen, da die von einer Benutzer*in durchgeführten Änderungen immer in einer Transaktion übernommen werden. Dies führt dazu, dass nach der Speicherung immer ein konsistenter Datensatz in der Datenbank vorhanden ist und es keine ggf. inkonsistenten Mischungen aus beiden Änderungen gibt.

Tabelle 4.3 Anwendungsfall: Bearbeiten von Datensätzen

Anwendungsfall	**Datensatz anlegen**
Akteure	Datenpfleger*innen, Administrator*innen
Vorbedingung	Benutzer*in angemeldet.
Nachbedingung Erfolgsfall	Der Datensatz wurde angelegt.
Ausnahmen / Fehlerfälle	
Nachbedingung Fehlerfall	
Ablauf	1. Die Benutzer*in wählt die Schaltfläche »Neues Substantiv«. 2. Es wird ein leeres Formular (identisch zu dem aus Tabelle 4.3 *Anwendungsfall: Bearbeiten von Datensätzen*) angezeigt. 3. Die Benutzer*in trägt die Daten ein. 4. Das Formular wird abgesendet. 5. Die Benutzer*in erhält Punkte für den neuen Datensatz und wird hierüber informiert (siehe Abschnitt 4.1.4 *Gamification*).
Varianten / Erweiterungen	• Es soll ein Verb, Adjektiv oder eine sonstige Wortart angelegt werden – dies findet analog statt und unterscheidet sich nur in Schritt 1 sowie den Feldern des Formulars aus Schritt 2 entsprechend der zu der entsprechenden Wortart zu erfassenden Daten.
Anmerkungen	

Tabelle 4.4 Anwendungsfall: Datensatz anlegen

Anwendungsfall	**Masseneditieren mehrer Datensätze**
Akteure	Administrator*innen
Vorbedingung	• Benutzer*in mit administrativen Rechten angemeldet. • (mehrere) Datensätze in der Datenbank vorhanden.
Nachbedingung Erfolgsfall	Alle gewählten Datensätze wurden wie gewünscht bearbeitet.
Ausnahmen / Fehlerfälle	Ein oder mehrere gewählte Datensätze erfüllen bereits die gewünschte Änderung.
Nachbedingung Fehlerfall	Die übrigen gewälten Datensätze werden angepasst und die beschriebenen Datensätze bleiben unverändert.
Ablauf	1. Die zu bearbeitenden Datensätze werden aus der Übersicht (eingrenzbar gemäß Tabelle 4.1 *Anwendungsfall: Datenabfrage mit Suche*) ausgewählt. 2. Es wird ein zusätzliches Auswahlfeld eingeblendet über welches die zu ändernde Eigenschaft ausgewählt werden kann. 3. Die Benutzer*in wählt die zu ändernde Eigenschaft aus. 4. Es wird ein weiteres Eingabefeld zur Angabe des neuen bzw. zu ergänzenden Wertes eingeblendet. 5. Die Benutzer*in gibt in dem zweiten Eingabefeld die Änderung an. 6. Das Formular wird abgesendet. 7. Die Änderungen werden gespeichert.
Varianten / Erweiterungen	
Anmerkungen	Da eine Massenoperation nur für Administratoren*innen des Systems verfügbar sein soll, wird auf eine Punktevergabe (siehe Abschnitt 4.1.4 *Gamification*) verzichtet.

Tabelle 4.5 Anwendungsfall: Masseneditieren mehrer Datensätze

Anwendungsfall	**Löschen eines Datensatzes**
Akteure	Administrator*innen
Vorbedingung	• Benutzer*in mit administrativen Rechten angemeldet. • Datensätze in der Datenbank vorhanden.
Nachbedingung Erfolgsfall	• Der geforderte Datensatz wurde gelöscht. • Alle Verknüpfungen zu diesem Datensatz wurden gelöscht.
Ausnahmen / Fehlerfälle	
Nachbedingung Fehlerfall	
Ablauf	1. Der zu löschende Datensatz wird aus der Übersicht (eingrenzbar gemäß Tabelle 4.1 *Anwendungsfall: Datenabfrage mit Suche*) zum Löschen ausgewählt. 2. Die Benutzer*in bestätigt die Löschung. 3. Der Datensatz wurde gelöscht.
Varianten / Erweiterungen	
Anmerkungen	Der Bestätigungsschritt wird implementiert, um versehentliche Löschungen zu verhindern, da keine »Undo«-Funktionalität vorgesehen ist.

Tabelle 4.6 Anwendungsfall: Löschen eines Datensatzes

4.3 Systemarchitektur

Die grundlegende Struktur geht aus dem MVC-Entwurfsmuster nach Abschnitt 2.2 hervor, welches in Rails als Basis erzwungen wird. Wie bereits in Abschnitt 2.2.2 *MVC im Web* als Fallstrick angedeutet, greift auch Rails auf Helper-Klassen zurück, um weiteren Quellcode auszulagern. Die Struktur des Projektes in Ruby on Rails wird daher auf einem Controller, einem Model und einem Ordner (mit diversen Template-Dateien für Teilbereiche der Anzeige, vgl. Abschnitt 2.3.1 *Ruby on Rails*) pro Datenklasse (vgl. Abschnitt 4.4 *Datenmodell*) und weiteren Helper-Klassen strukturiert nach Aufgabengebiet bestehen. In Abschnitt 5.2 *Implementation* wird ein Einblick in einige dieser Komponenten gegeben.

4.4 Datenmodell

Zur Umsetzung des Datenmodells wurde sich auf eine Teilmenge der gemäß Abschnitt A.9.1 *Datenbankstruktur, Initialversion* geforderten (Meta-)Daten beschränkt, die sich in dem folgenden Klassendiagramm wiederfinden. Die Bedeutungen einzelner Attribute werden im Folgenden noch zusätzlich erleutert und ggf. weitere Einschränkungen der enthaltenen Daten dokumentiert. Es wird im Klassendiagramm auf die Darstellung von in Rails automatisch vorhandenen Getter- und Setter-Methoden für die definierten Attribute verzichtet, um die Diagramme nicht unnötig zu überladen. Es werden somit nur Methoden aufgeführt, welche über dieses Verhalten hinaus eine zusätzliche Funktion erbringen sollen.

Grundlegend ist hierbei eine gemeinsame Oberklasse für alle Wortarten `Word` vorgesehen, welche die gemeinsamen Attribute (wie z. B. *name* als Grundform des Wortes, *syllables* zur Darstellung der Silben oder *meaning* für eine Kurzbedeutung, um bspw. Teekesselchen unterscheiden zu können) aufnimmt. Davon abgeleitet sind die Subklassen `Noun`, `Verb` und `Adjective` welche für die jeweiligen Wortarten individuellen Attribute ergänzen.

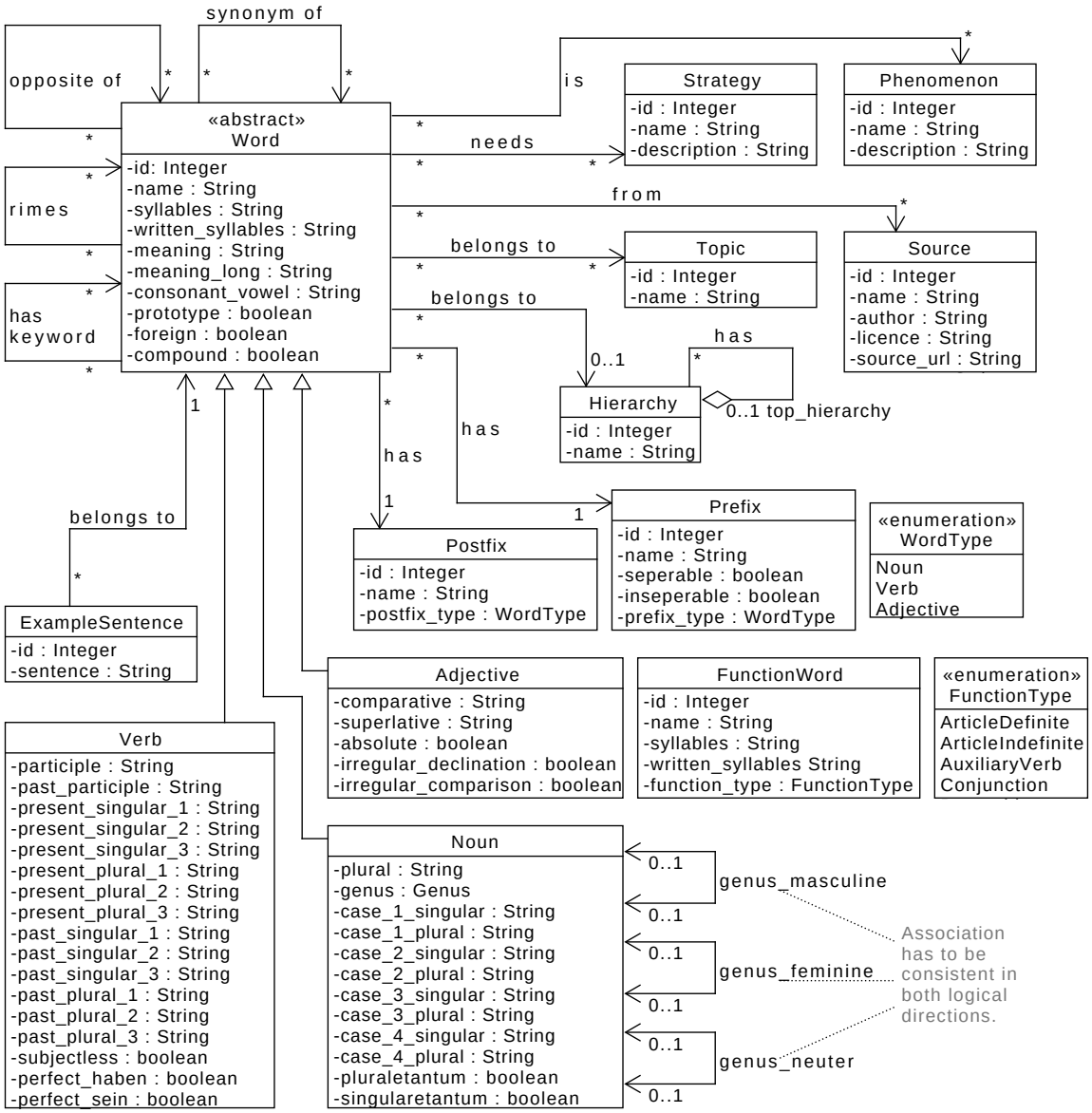

Abbildung 4.1 Klassendiagram aller Datenklassen

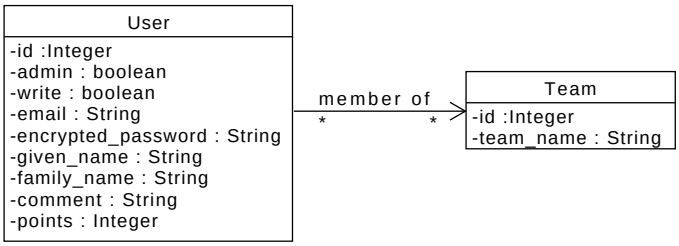

Abbildung 4.2 Klassendiagram der zur Accountverwaltung gehörigen Klassen

Attribut	Bedeutung im Anwendungskontext	Nebenbedingungen
name	Grundform des Wortes	
syllables	Sprechsilben	String, getrennt über `-`
written_syllables	Schreibsilben	String, getrennt über `|`
meaning	Kurzbedeutung (z. B. zur Unterscheidung von Teekesselchen)	
meaning_long	Beschreibungstext bzw. Erklärung	
consonant_vowel	Konsonant-Vokal-Folge	*generiert*, String basierend auf Grundform: Vokale (inkl. ä, ö, ü) werden durch `V`, Konsonanten durch `K` ersetzt
prototype	Modellwort, Wörter welche ein einzelnes Rechtschreibphänomen beinhalten	
foreign	Fremdwörter	
compound	Wortzusammensetzung	

Tabelle 4.7 Attributerklärung zur Klasse `Word`

Attribut	Bedeutung im Anwendungskontext	Nebenbedingungen
comparative	Komperativ, 1. Steigerung	
superlative	Superlativ, 2. Steigerung	
absolute	Absolutadjektiv, keine Steigerung möglich	
irregular_declination	Unregelmäßige Deklination (bspw. »hohes Haus« statt »hoches Haus«)	
irregular_comparision	Unregelmäßige Steigerung (bspw. »gut, besser, am besten«)	

Tabelle 4.8 Attributerklärung zur Klasse `Adjective`

Attribut	Bedeutung im Anwendungskontext	Nebenbedingungen
plural	Plural	
genus	Grammatisches Geschlecht	Kann `Famininum`, `Maskulinum`, `Neutrum` oder eine beliebige Kombination sein (bspw. »der/die Abgeordnete«)
case_x_{singular/plural}	x-ter Fall Singular bzw. Plural	
pluraletantum	Pluralwort, kein Singular möglich (bspw. »Ferien«)	
singularetantum	Singularwort, kein Plural möglich (bspw. »Adel«)	

Tabelle 4.9 Attributerklärung zur Klasse `Noun`

Attribut	Bedeutung im Anwendungskontext	Nebenbedingungen
pariciple	Partizip	
past_participle	Partizip Perfekt	
{present/past}_{singular/plural}_x	x-te Person Singular bzw. Plural im Präsenz bzw. Präteritum	
subjectless	Subjektloses Verb	
perfect_{haben/sein}	Perfektbildung über »haben« bzw. »sein«	

Tabelle 4.10 Attributerklärung zur Klasse `Verb`

Für die Accountverwaltung (inklusive einer Möglichkeit zur Verwaltung von Teams) werden weitere Klassen benötigt. Das Klassendiagramm aus Abbildung 4.2 auf Seite 28 zeigt die dafür benötigten Klassen `User` und `Team`. Abhängig von der konkreten Implementation können noch weitere Attribute ergänzt werden. Wie bereits zuvor wurde auch hier auf eine Darstellung der automatisch vorhandenen Getter und Setter verzichtet.

Kapitel 5

Systemumsetzung

Die Systemumsetzung basiert auf verschiedenen Werkzeugen, welche in der Systementwicklungsumgebung zum Einsatz kommen und setzt hierbei insbesondere auf Ruby on Rails auf. In Abschnitt 5.2 *Implementation* wird auf einige Elemente des Komplettsystems eingegangen und dabei auch ein direkter Einblick in die Implementation geboten. Da das System auch bereits während der Entstehung dieser Arbeit im Produktivbetrieb zum Einsatz gekommen ist, wird ebenfalls auf die Benutzbarkeit und Bedienbarkeit in Abschnitt 5.3 geachtet.

5.1 Systementwicklungsumgebung

Zur Umsetzung des Systems wird auf unterschiedliche Komponenten und Werkzeuge aufgebaut, welche im Entwicklungssystem (vgl. Abschnitt 5.1.1) und Test- bzw. Produktivsystem (vgl. Abschnitt 5.1.2) zum Einsatz kommen.

5.1.1 Entwicklungssystem

Die Entwicklung findet auf einem System mit Manjaro-Linux statt. Als Datenbank wird MariaDB eingesetzt, nachdem SQLite in der Datenhaltung verwendete Funktionen nicht unterstützt. Da das System auf Rails basiert wird *bundler*[1] und *rake*[2] eingesetzt und insbesondere auch die `rails console` zum leichteren Debugging des Systems. Verwaltet wird der Quellcode in einem Git-Repository im Gitlab des FB 3[3]. Sofern an einzelnen Gems anpassungen notwendig werden (bspw. auf Grund von veralteten bzw. fehlerbehafteten Versionen) werden diese als Fork auf Github[4] erzeugt und verwaltet. Die gesamte Entwicklungsarbeit findet dabei konsolenbasiert mit `tmux` und in `vim` als Texteditor statt, außerdem werden

[1] *https://rubygems.org/gems/bundler*, abgerufen am: 17.07.2019
[2] *https://rubygems.org/gems/rake*, abgerufen am: 17.07.2019
[3] *https://gitlab.informatik.uni-bremen.de/word/wordapp*, abgerufen am: 07.06.2019
[4] *https://github.com/*, abgerufen am: 07.06.2019

die Entwicklertools von Firefox verwendet. Das System ist hauptsächlich in und für Firefox entwickelt und darauf getestet, es wird zur Überprüfung der kompatibilität jedoch auch in Chrome bzw. Chromium getestet.

5.1.2 Test- und Produktivsystem

Für den Einsatz des Systems durch Prof. Dr. Frank J. Müller ist ein Produktivsystem auf einer Virtuellen Maschine des Fachbereich 3 eingerichtet worden. Hierbei wird ein Debian Stable (9, *stretch*) als Grundsystem verwendet. Die Datenbank selbst wird, wie im Entwicklungssystem, mit MariaDB realisiert, wobei mittels Cronjob ein tägliches Backup der gesamten Datenbank erzeugt wird. Um die Rails-Anwendung auszuführen werden die Apache-Module *Phusion Passenger*[5] und für die Nutzung von Shibboleth *shib2*[6] mit einem Apache-Webserver eingesetzt.

Eine Testinstanz mit identischen Komponenten wird auf einem separaten System eingerichtet, um frühe Entwicklungsversionen unter realitätsnahen Bedingungen zu testen. Dieses System ist außerdem für die Testnutzung im Rahmen dieser Arbeit erreichbar unter *https://wortschatz.dennisschuerholz.de* gedacht:
Administrator*in-Account: `gutachter-admin@dennisschuerholz.de`,
normaler Account: `gutachter-benutzer@dennisschuerholz.de`,
lesender Account: `gutachter-leser@dennisschuerholz.de`,
Passwort: `Bachelorarbeit`.
Nach erfolgter Begutachtung der Arbeit werden diese Zugangsdaten widerrufen. Ein Login über Shibboleth ist in dieser Instanz nicht möglich.

5.2 Implementation

Für die Implementation des Systems wird auf die in den Grundlagen beschriebenen Web-Frameworks Rails (vgl. Abschnitt 2.3.1) und Bootstrap (vgl. Abschnitt 2.3.2) zurückgegriffen. In diesem Abschnitt werden Teilbereiche des Systems, wie sie auch in Abschnitt 4.2 *Anwendungsfälle* beschrieben werden, in ihrer visuellen Darstellung und Implementation auszugsweise eingearbeitet.

5.2.1 Punktesystem

Nach der Anmeldung im System wird die eigene Profilübersicht wie in Abbildung 5.1 *Startseite für angemeldete Benutzer*innen (eigenes Profil)* angezeigt. Hier werden die im System hinterlegten persönlichen Daten (Vorname, Nachname, E-Mailadresse), der Accountname (über Shibboleth oder bei lokalen Accounts identisch zur E-Mailadresse), die zugeordneten Zugriffsrechte (Administrator*in, Benutzer*in oder

[5] *https://de.wikipedia.org/wiki/Phusion_Passenger*, abgerufen am: 07.06.2019
[6] *https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPApacheConfig*, abgerufen am: 04.06.2019

Leser*in) sowie der initial ausgeblendete API-Token (vgl. Abschnitt 5.2.6 *Programmschnittstellen/API*) angezeigt. Das Anzeigebild (für die Benutzer*innenübersicht und immer auch in der Navigationsleiste ganz rechts) wird mit Hilfe der E-Mailadresse beim Dienst Gravatar[7] abgerufen. In der rechten Spalte, sowie in der Navigationsleiste, sind die im System erzielten Punkte durch Bearbeitung der Datenbasis zu sehen (vgl. Abschnitt 4.1.4 *Gamification*). Ebenfalls in der Navigationsleiste und somit in jeder View findet sich die »Wortartübergreifende Suche« über welche Benutzer*innen direkt in die Bearbeitungsansicht beliebiger Wörter gelangen.

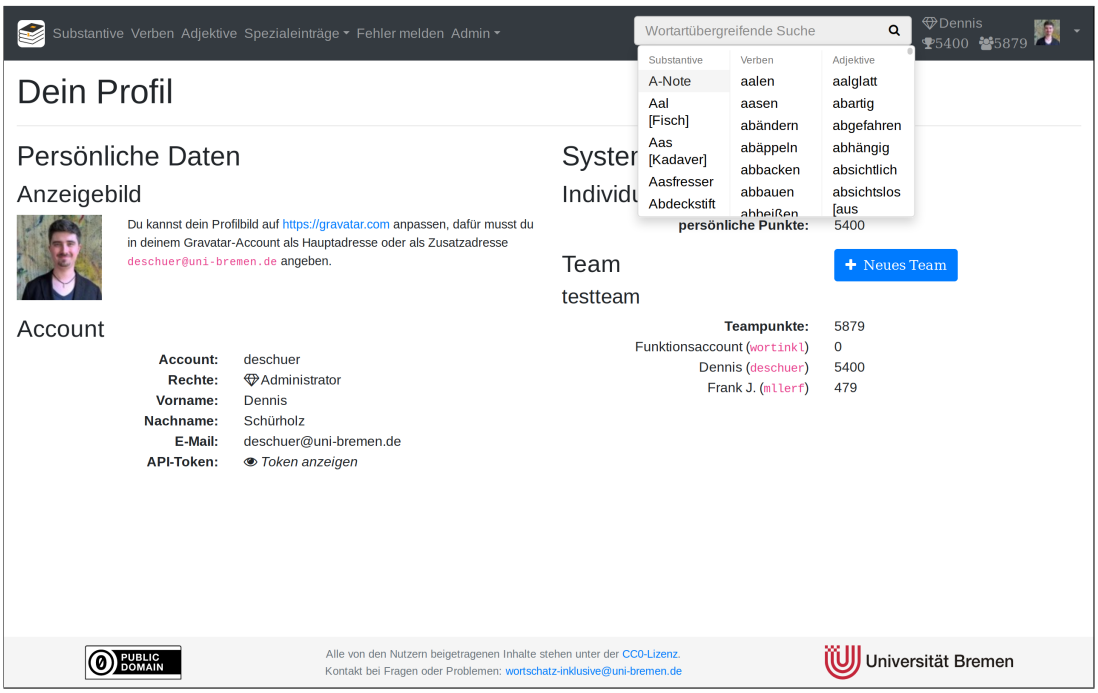

Abbildung 5.1 Startseite für angemeldete Benutzer*innen (eigenes Profil)

5.2.2 Listenansicht

Zu jeder Wortart existiert eine Übersicht in Form einer Listendarstellung, welche die jeweils wichtigsten Attribute darstellt. Dies entspricht der Tabelle 4.1 *Anwendungsfall: Datenabfrage mit Suche*, auf den Aspekt der Suche bzw. des Filterns wird im unteren Abschnitt eingegangen. Ein Beispiel für diese Darstellung für die Substantive ist in Abbildung 5.2 *Listenansicht der Substantive* abgebildet und ist unter `/nouns/` abrufbar. Da diese Darstellung für andere Wortarten äquivalent erfolgt und sich lediglich in den dargestellten Attributen unterscheidet, finden sich im Anhang als Abbildung A.1 und Abbildung A.3 Screenshots der Views für Verben bzw. Adjektive.

[7] *https://de.gravatar.com*, abgerufen am: 15.06.2019

Dabei werden manche Attribut(gruppen) nur über den Zustand »eingetragen« bzw. »nicht eingetragen« abgebildet (so im Beispiel die zum Subtantiv hinterlegten Formen in den unterschiedlichen Fällen). Diese Darstellung soll ein einfaches Auffinden von Lücken in der Datenbank erleichtern und so zur Mitarbeit an dem Projekt und somit der Vervollständigung der (Meta-)Daten anregen. In dieser Ansicht ist es für Administrator*innen ebenfalls möglich durch Markieren mehrerer Datensätze über die Kontrollkästchen (in der Tabelle links) ein Massenbearbeiten bestimmter Eigenschaften, wie in Tabelle 4.5 *Anwendungsfall: Masseneditieren mehrer Datensätze* beschrieben, auszulösen. Dies erleichtert Ergänzungen (z. B. von vorkommenden Rechtschreibphänomenen oder Reimwörtern) auf einer ggf. gefilterten Menge an Wörtern. Hierbei wird nach erfolgter Auswahl unter der Tabelle ein Formular mit mehreren Auswahlmenüs angezeigt um so die gewünschten Änderungen auf alle ausgewählten Datensätze anzuwenden (vgl. Funktion *bulk_update* in Quellcode 5.4 *Auszug aus* `app/controller/nouns_controller.rb`). Zusätzlich ist es Administrator*innen auch möglich über die Papierkorb-Symbole (in der Tabelle rechts) Wörter zu löschen wie in Tabelle 4.6 *Anwendungsfall: Löschen eines Datensatzes* vorgesehen, wenn diese bspw. mehrfach vorkommen, ohne eine abweichende Bedeutung zu haben.

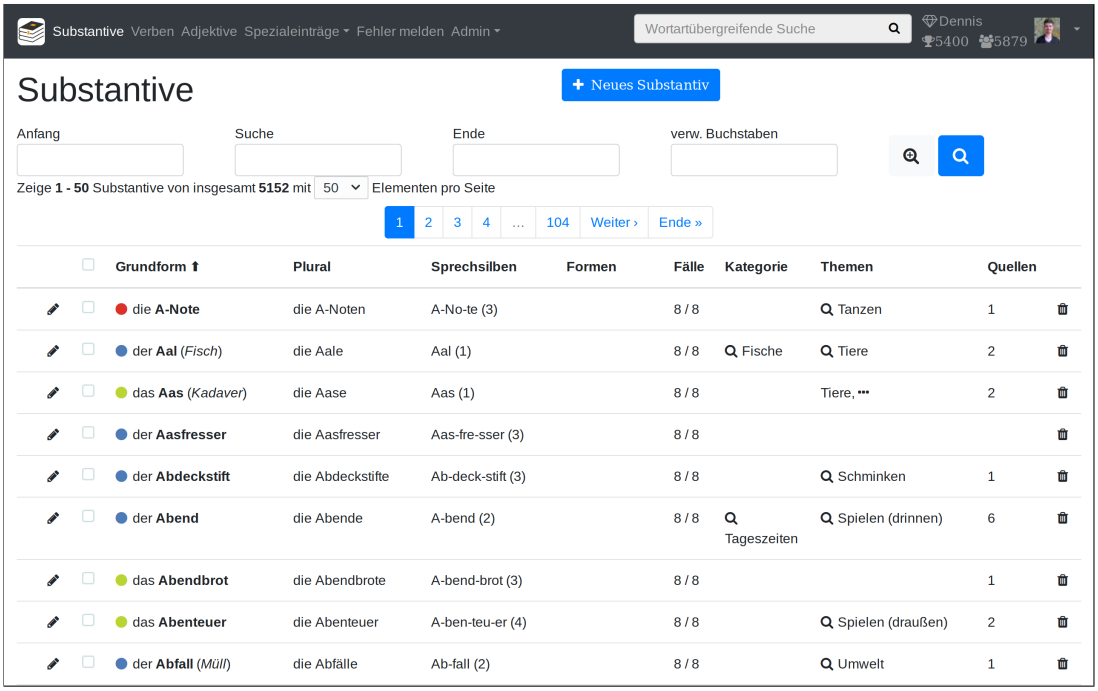

Abbildung 5.2 Listenansicht der Substantive

Um diese Listen nach frei wählbaren Kriterien eingrenzbar zu gestalten, werden der Benutzer*in verschiedene Möglichkeiten zur Filterung gegeben. In Abbildung 5.3 *Filteroptionen der Substantive* sind die für alle Wortarten identischen Suchschlüssel sowie die individuellen Möglichkeiten für Substantive aufgezeigt. Äquivalent verfügen auch die Verben und Adjektive über individualisierte Suchfilter. Diese können ebenfalls im Anhang unter Abbildung A.2 und Abbildung A.4 begutachtet werden.

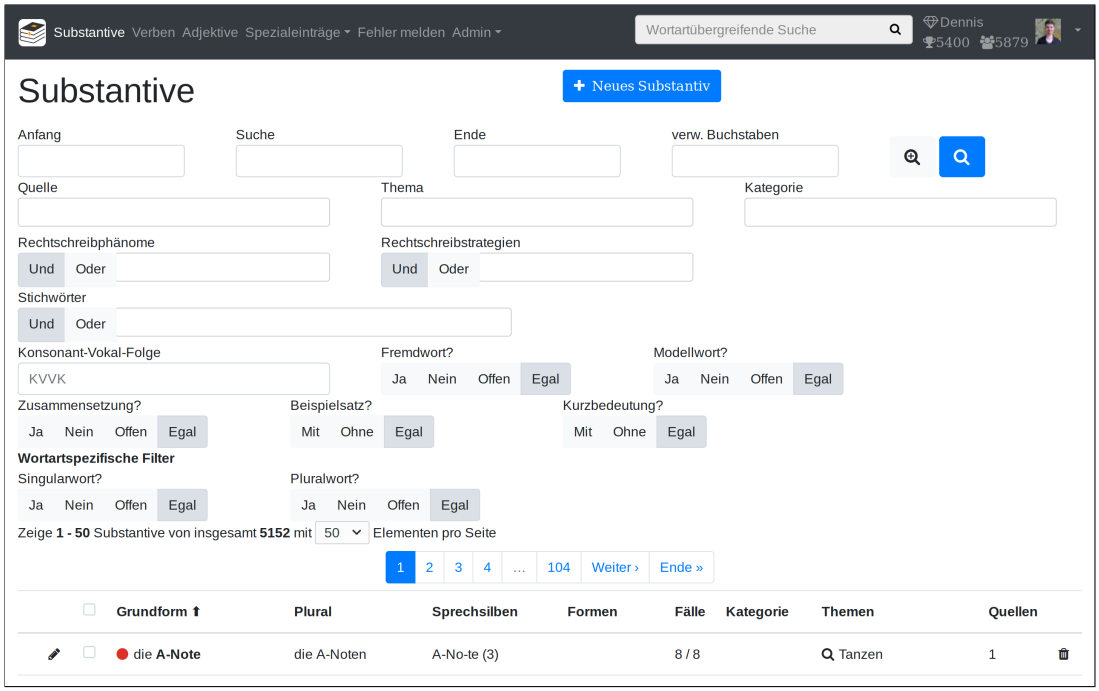

Abbildung 5.3 Filteroptionen der Substantive

Realisert wird die Suche über das Gem *filterrific*[8], welches Möglichkeiten zum Filtern und Sortieren von Datensätzen über spezielle Links bzw. Formulare anbietet. Exemplarisch sei in Quellcode 5.1 *Auszug aus `lib/word_filter.rb`* ein Ausschnitt des Quelltextes für die Umsetzung der Filter gegeben. Solche Filter können sowohl über interne Mechanismen von ActiveRecord wie in `joins(:word).where(foreign↩ : value)` (Zeile 172 in Quellcode 5.1) als auch über selbstkonstruiertes SQL wie in `where("LOWER (#{↩ table_name}.name)LIKE ? COLLATE utf8_bin", term)` (Zeile 98 in Quellcode 5.1) umgesetzt werden. Diese Varianten können ebenfalls in Kombination eingesetzt werden wie in Zeile 122 aus Quellcode 5.1 zu sehen. Über den Namen des Scopes[9] bspw. `search_wordstarts` (Zeile 73 in Quellcode 5.1) wird die Zuordnung von Formularfeld bzw. Parameter in der URL auf den anzuwendenden Filter vorgenommen (vgl. Tabelle 5.3 *Übersicht einiger Suchschlüssel*). Ein Ausschnitt des Quelltextes für die Anzeige des Filter-Formulars kann dem Quellcode 5.2 *Auszug aus `app/views/shared/_filter.html.haml`* entnommen werden. Die Anwendung bzw. Aktivierung des Filters findet in dem entsprechenden Controller, wie bspw. für die Substantive in Quellcode 5.4 *Auszug aus `app/controller/nouns_controller.rb`* zu finden, statt.

[8] *https://rubygems.org/gems/filterrific*, abgerufen am: 17.07.2019
[9] *https://guides.rubyonrails.org/active_record_querying.html#scopes*, abgerufen am: 24.05.2019

```
73  scope :search_wordstarts, lambda { |search|
74    return nil if search.blank?
75    search_wordquery(search + '%')
76  }
        ⋮
95  scope :search_wordquery, lambda { |search|
96    return nil if search.blank?
97    term = search.downcase.gsub('*', '%').gsub('?', '_')
98    where("LOWER(#{table_name}.name) LIKE ? COLLATE utf8_bin", term)
99  }
        ⋮
104 scope :with_topic, lambda { |topic|
105   joins(:word).joins("INNER JOIN `topics_words` ON `words`.`id` = `topics_words`.`word_id`").where(topics_words: { topic_id: topic})
106 }
        ⋮
126 scope :with_keywords, lambda { |keywords|
127   return if keywords.keywords.nil?
128   keywords.keywords.select!{|w| /^[0-9]+$/.match? w}
129   return nil if keywords.keywords.empty?
130   count = ""
131   count = "COUNT(*) = #{keywords.keywords.length}" if keywords.conjunction == "AND"
132   joins(:word).joins("INNER JOIN `keywords` ON `words`.`id` = `keywords`.`word_id`").where(keywords: { keyword_id: keywords.keywords }).group('`keywords`.`word_id`').having(count).distinct
133 }
        ⋮
170 scope :is_foreign, lambda { |cb|
171   value = cb == "CHECKED" ? true : (cb == "UNCHECKED" ? false : nil)
172   joins(:word).where(foreign: value) unless cb == 'EMPTY'
173 }
```

Quellcode 5.1 Auszug aus `lib/word_filter.rb`

```
16  = form_for_filterrific @filterrific do |f|
17    .row.align-items-end
18      .col
19        .row.col= t('filter.wordstarts')
20        .row.col
21          = f.text_field(:search_wordstarts, class: 'form-control filterrific-periodically-observed')
                ⋮
```

```
52        .col
53          .row.col= t('filter.topic')
54          .row.col
55            = f.select(:with_topic,
56              Topic.all.order(:name).map{|s|[s.name, s.id]},
57              { include_blank: true, selected: @filterrific.with_topic },
58              { class: "form-control selectize-select"})
                  ⋮
101       .col
102         .row.col= t('filter.keywords')
103         .row.col.input-group
104           = f.fields_for :with_keywords do |with_keywords|
105             .input-group-prepend.btn-group.btn-group-toggle{ 'data-toggle' => '↩
                    buttons' }
106               - (['AND', 'OR'].map{|e| [t("filter.and_or.short.#{e}"), e]}).↩
                      each do |rb|
107                 - checked = @filterrific.with_keywords.try(:conjunction) == rb↩
                        [1] || @filterrific.with_keywords.try(:conjunction).nil? &&↩
                         rb[1] == 'AND'
108                 = with_keywords.label :conjunction,
109                   { class: "btn btn-light#{checked ? ' active' : ''}" } do
110                   = with_keywords.radio_button(:conjunction,
111                     rb[1],
112                     checked: checked)
113                   = rb[0]
114             = with_keywords.select(:keywords,
115               [],
116               { include_blank: true, selected: @filterrific.with_keywords.try(:↩
                      keywords) },
117               { class: "form-control selectize-select optgroup-select", ↩
                      multiple: true, 'data-lazy-load-url' => all_short_words_path,↩
                       'data-selected'=> @filterrific.with_keywords.try(:keywords).↩
                      to_json })
                    ⋮
123       = filter_checkbox_field(f, @filterrific, :is_foreign, t('filter.foreign')↩
            )
```

Quellcode 5.2 Auszug aus `app/views/shared/_filter.html.haml`

```
2     def filter_checkbox_field(form, filterrific, name, label)
3       checkbox_field(form, name, label, 'filter.checkboxes.short', ['CHECKED', '↩
          UNCHECKED', 'NONE', 'EMPTY'], filterrific)
4     end
5     def inline_checkbox_field(form, name, t_keybase, t_keys, params)
6       capture_haml do
7         haml_tag(:div, class: 'btn-group btn-group-toggle', 'data-toggle' => '↩
            buttons') do
8           (t_keys.map{|e| [t("#{t_keybase}.#{e}").html_safe, e]}).each do |rb|
9             checked = (params.try(name) || params.try(:[], name)) == rb[1] || (↩
                params.try(name) || params.try(:[], name)).blank? && rb[1] == '↩
                EMPTY'
10            haml_concat(form.label(name,
11              { class: "btn btn-light#{checked ? ' active' : ''}" }) do
12              capture_haml do
13                haml_concat form.radio_button(name,
14                  rb[1],
15                  checked: checked)
16                haml_concat rb[0]
17              end.html_safe
18            end)
19          end
20        end
21      end
22    end
23    def checkbox_field(form, name, label, t_keybase, t_keys, params)
24      capture_haml do
25        haml_tag :div, class: 'col-12 col-sm-6 col-md-4 col-lg-3 col-xl-2' do
26          haml_tag :div, class: 'row col' do
27            haml_concat label
28          end
29          haml_tag :div, class: 'row col' do
30            haml_concat inline_checkbox_field(form, name, t_keybase, t_keys, ↩
                params)
31          end
32        end
33      end
34    end
        ⋮
117   def input_lazy_select(form, name, path, label, text=nil, explanation=nil, ↩
        selected={}, json_attribute='words', multiple=false, disabled=false, ↩
        optgroup = false, hsh = {})
118     capture_haml do
```

```
119       haml_tag :div, class: 'row form-group' do
120         haml_tag :div, class: 'col-sm-2 text-right-not-xs form-control-↩
              plaintext' do
121           if text.nil?
122             haml_concat form.label name, label.html_safe
123           else
124             haml_concat form.label name, label, class: 'sr-only'
125             haml_concat text.html_safe
126           end
127         end
128         haml_tag :div, class: 'col-sm-10' do
129           haml_concat form.select name,
130             [],
131             { include_blank: true, selected: selected },
132             { class: "form-control selectize-select#{' optgroup-select' if ↩
                  optgroup}", multiple: multiple, disabled: disabled , 'data-lazy↩
                  -load-url' => path, 'data-selected' => selected, 'data-lazy-↩
                  load-attribute' => json_attribute }.merge(hsh)
133         end
134         explain(explanation, disabled)
135       end
136     end
137   end
```

Quellcode 5.3 Auszug aus `app/helpers/form_helper.rb`

```
8  def index
9    @filterrific = initialize_filterrific(
10     Noun,
11     params[:filterrific],
12   ) or return
13   @nouns = @filterrific.find.page(params[:page]).per(current_user.settings.↩
       page_size)
14 end
     ⋮
40 def bulk_update
41   if params[:word_ids].is_a?(Array) && params[:word_ids].length > 1
42     @nouns = Noun.find(params[:word_ids])
43     @nouns.each do |noun|
44       noun.update(bulk_params(noun))
45     end
46     redirect_to nouns_path, notice: t('notice.bulk_edit_saved', elements: ↩
         @nouns.size)
47   else
```

```
48      redirect_to nouns_path
49    end
50  end
```

Quellcode 5.4 Auszug aus `app/controller/nouns_controller.rb`

5.2.3 Bearbeitungsansicht

In Tabelle 4.3 *Anwendungsfall: Bearbeiten von Datensätzen* wird das Bearbeiten und in Tabelle 4.4 *Anwendungsfall: Datensatz anlegen* das Anlegen eines Datensatzes addressiert. Dieses Bearbeiten bzw. Anlegen eines Datensatzes findet über eine simple Eingabemaske mit allen hinterlegten Feldern und Verknüpfungsmöglichkeiten statt. Die Auswahlfelder für die Wortverknüpfungen werden ebenso wie die äquivalenten Felder in der Filterfunktion aus Abschnitt 5.2.2 erst nach dem Anzeigen der Seite per JavaScript nachgeladen um die initiale Seitenlade- bzw. Seitenaufbauzeit möglichst kurz zu halten (vgl. Abschnitt 5.2.6 *Programmschnittstellen/API*). Für diese Auswahlelemente wird auf das Gem *selectize-rails*[10] zurückgegriffen, welches *selectize.js* (ein Alternativframework zu den bekannteren *chosen* bzw. *Select2*) verwendet. Einen Auszug aus der View sowie den dazugehörigen Helper- und JavaScript-Quellcodes kann in Quellcode 5.5 *Auszug aus* ***app/views/nouns/_form.html.haml*** bzw. Quellcode 5.3 *Auszug aus* ***app/helpers/form_helper.rb*** (ab Zeile 117) und Quellcode 5.6 *Auszug aus* ***app/assets/javascripts/initializers.coffee*** gefunden werden.

```
65        .col-md-6
66          = input_lazy_select form,
67            :genus_neuter_id,
68            all_short_nouns_path('filterrific[with_genus]' => 'NEUTER'), 'Form&↩
                  nbsp;(n)', nil, nil, @noun.genus_neuter_id, 'nouns', false, ↩
                  local_assigns[:edit].nil?
69      .row
70        .col-md-6
71          = input_lazy_select form,
72            :genus_masculine_id,
73            all_short_nouns_path('filterrific[with_genus]' => 'MASCULINUM'), '↩
                  Form (m)', nil, nil, @noun.genus_masculine_id, 'nouns', ↩
                  false, local_assigns[:edit].nil?
74        .col-md-6
75          = input_lazy_select form,
76            :genus_feminine_id,
77            all_short_nouns_path('filterrific[with_genus]' => 'FEMININUM'), 'Form↩
                   (f)', nil, nil, @noun.genus_feminine_id, 'nouns', false, ↩
                  local_assigns[:edit].nil?
```

Quellcode 5.5 Auszug aus `app/views/nouns/_form.html.haml`

[10] *https://rubygems.org/gems/selectize-rails*, abgerufen am: 17.07.2019

```
31 $('select[data-lazy-load-url].selectize-select').each (index, element) ->
32   $(element).selectize
33     plugins: ['remove_button', 'optgroup_columns']
34     preload: true
35     maxOptions: 1000
36     selectOnTab: true
37     valueField: 'id'
38     labelField: 'name'
39     searchField: 'name'
40     optgroupField: 'actable_type'
41     optgroupLabelField: 'name'
42     optgroupValueField: 'type'
43     optgroups: [
44       {type: 'Noun', name: t.selectize.nouns}
45       {type: 'Verb', name: t.selectize.verbs}
46       {type: 'Adjective', name: t.selectize.adjectives}
47     ]
48     optgroupOrder: ['Noun', 'Verb', 'Adjective']
49     lockOptgroupOrder: true
50     load: (query, callback) ->
51       if (query.length)
52         return callback()
53       $.ajax
54         url: $(element).data('lazyLoadUrl')
55         type: 'GET'
56         error: () ->
57           callback()
58         success: (res) ->
59           callback(res.words)
60           for sel in $(element).data('selected')
61             $(element)[0].selectize.addItem(sel)
```

Quellcode 5.6 Auszug aus `app/assets/javascripts/initializers.coffee`

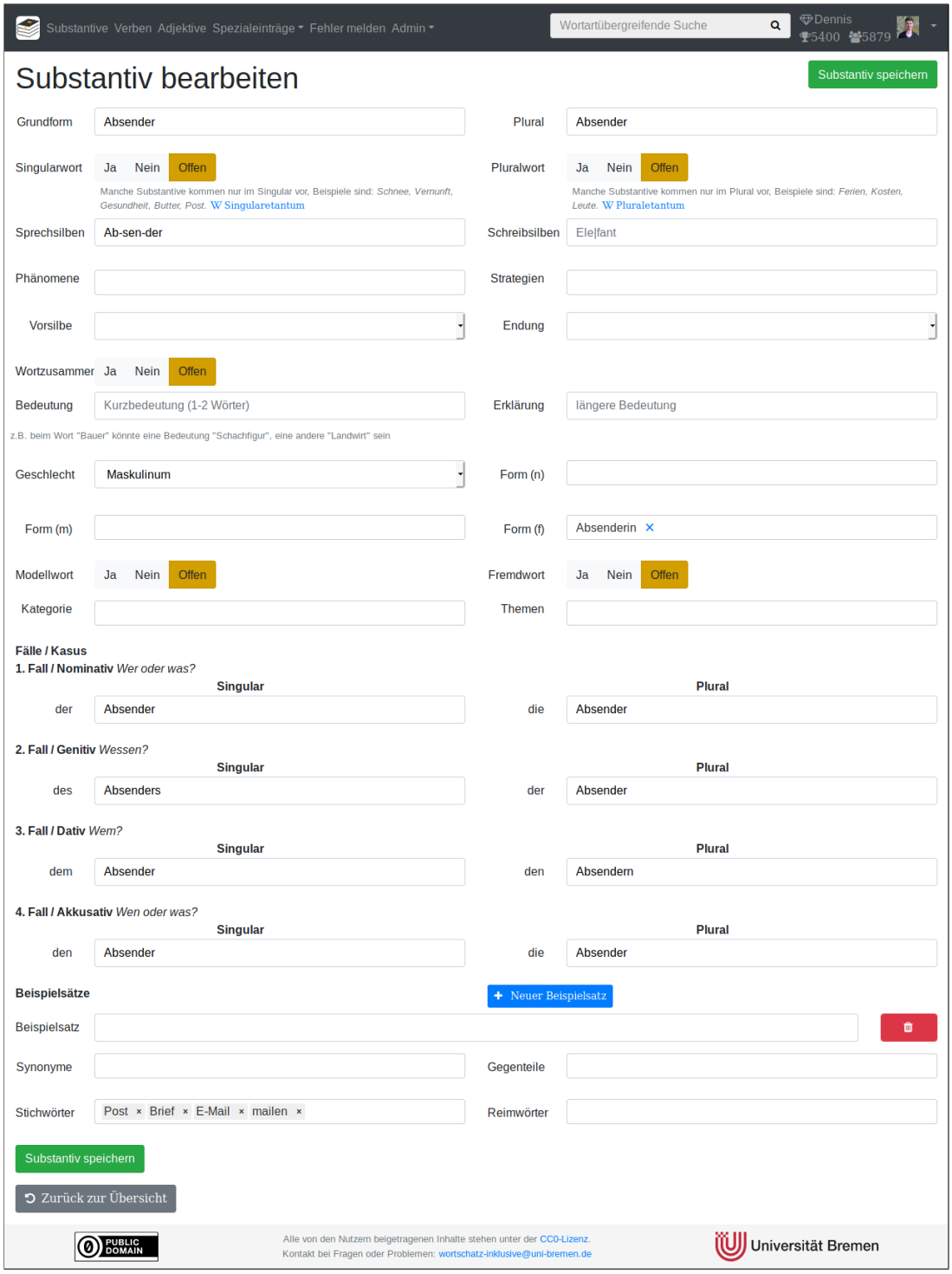

Abbildung 5.4 Bearbeitungsansicht der Substantive

5.2.4 Quellenimport

Beim Importieren von Quellen in Form von Wortansammlungen (wie bspw. dem Bremer Rechtschreibschatz von 2015[11]) wird das Formular aus Abbildung 5.5 *Quellenimport* verwendet um auch hierbei interessante Metadaten zum Quellmaterial erfassbar zu machen, so wie es auch in Tabelle 4.2 *Anwendungsfall: Import einer Datensammlung als CSV* beschrieben ist. Dabei werden als Quellen CSV-Dateien verwendet, welche Einträge der Form `Wortart,Wort,Bedeutung,Themen,Genus` enthalten. Die letzten drei Felder sind dabei optional, das Feld `Themen` kann sowohl als Auflistung (mit `;` getrennt oder als einzelnes Feld definiert werden, das Genus findet nur bei Substantiven anwendung.

Mehrere Einträge gelten (auch zu bereits in der Datenbank vorhandenen) als identisch, wenn sie der selben Wortart angehören, das selbe Wort sind bzw. die selbe Grundform haben und die identische Kurzbedeutung (um Teekesselchen auszuschließen). Solche Einträge werden miteinander verknüpft und die zugeordneten Themen akkumuliert. Gibt die importierte Quelle keine Bedeutung an werden für den Vergleich mit der Datenbank lediglich Wortart und Grundform berücksichtigt, da keine (bereits vorhandene) Bedeutung ausgeschlossen werden kann. Diese Zuordnung zu bereits in der Datenbank vorhandenen Wörtern kann dem Quellcode 5.7 *Auszug aus `app/models/source.rb`* ab Zeile 32 entnommen werden.

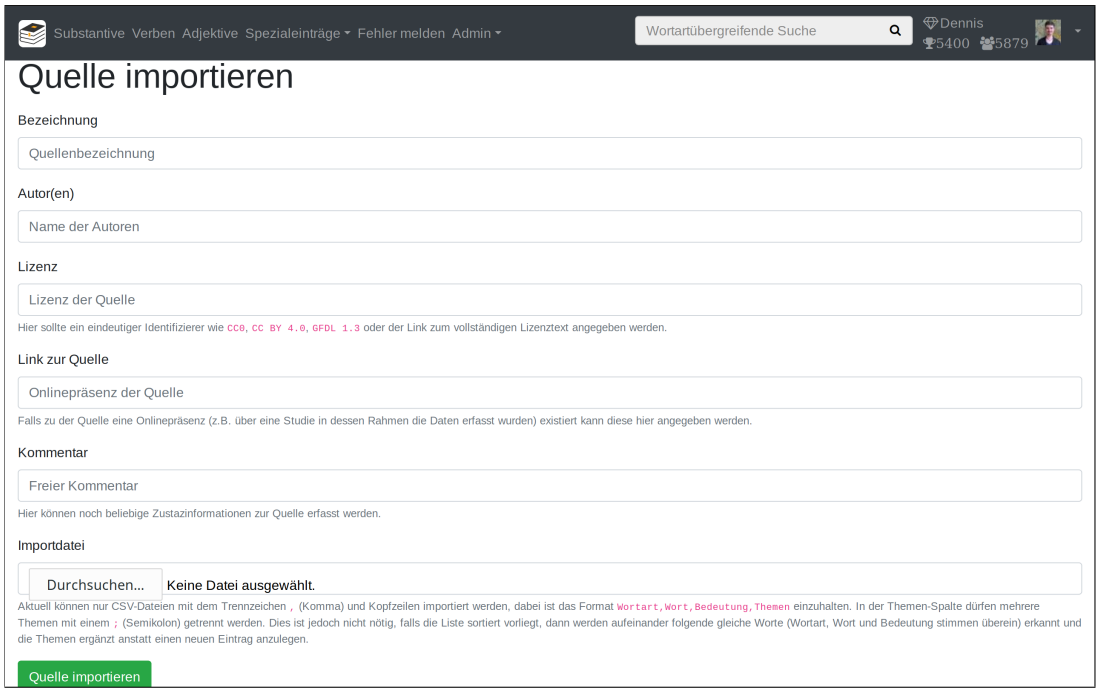

Abbildung 5.5 Quellenimport

[11] *https://www.lis.bremen.de/va/detail.php?gsid=bremen56.c.84913.de*, abgerufen am: 04.06.2019

```
15  CSV.foreach(file.path, headers: true, col_sep: ',', encoding: detection[:↩
        encoding]) do |row|
16    return nil if word_type(row[0]).nil? or (row[1].nil? or row[1].empty?)
17    hash = Hash.new
18    hash[:type] = word_type row[0]
19    hash[:data] = Hash.new
20    hash[:data][:name] = row[1].strip
21    hash[:data][:meaning] = row[2].strip unless row[2].nil?
22    hash[:data][:genus_id] = genus row[4] if hash[:type] == :noun
23    last = data.last
24    if !last.nil? and last[:data][:name] == hash[:data][:name] and last[:data][:↩
          meaning] == hash[:data][:meaning] and last[:type] == hash[:type]
25      last[:topics].concat (row[3]||'').split(';')
26    else
27      hash[:topics] = (row[3]||'').split(';')
28      data << hash
29    end
30  end
31  data.each do |w|
32    if w[:data][:meaning].blank?
33      words = w[:type].to_s.classify.constantize.where(name: w[:data][:name])
34    else
35      words = [w[:type].to_s.classify.constantize.find_by(name: w[:data][:name], ↩
            meaning: w[:data][:meaning])]
36    end
37    words = [nil] if words.count == 0
38    words.each do |word|
39      if word.nil?
40        word = w[:type].to_s.classify.constantize.create! w[:data]
41        count += 1
42      else
43        if w[:type] == :noun
44          word.genus_id ||= w[:data][:genus_id]
45        end
46        updated += 1
47      end
48
49      w[:topics].uniq.each do |t|
50        begin
51          word.topics << Topic.find_or_create_by(name: t.strip)
52        rescue ActiveRecord::RecordNotUnique
53          logger.debug "Duplicate topic '#{t.strip}' for #{w[:type]} '#{word.name↩
              }'"
```

```
54        end
55      end
56      begin
57        word.sources << source
58      rescue ActiveRecord::RecordNotUnique
59        updated -= 1
60        logger.debug "Duplicate source '#{source.name}' for #{w[:type]} '#{word.↩
              name}'"
61      end
62    end
63  end
```

Quellcode 5.7 Auszug aus `app/models/source.rb`

```
29  def create
30    if params[:source][:file].nil?
31      redirect_to new_source_path, alert: 'Es wurde keine Quelldatei ausgewählt.'
32      return
33    end
34    file = params[:source].delete :file
35    @source = Source.new(source_params)
36
37    if @source.save
38      count = Source.import(file, @source)
39      if count.nil?
40          @source.delete
41          redirect_to new_source_path, alert: "Keine Wörter importiert, Fehler in↩
                der Quelldatei \"#{file.original_filename}\"."
42      else
43          redirect_to sources_path, notice: "#{count[:new]} neue Wörter ↩
                importiert und #{count[:updated]} vorhandene Wörter verknüpft aus ↩
                der Datei \"#{file.original_filename}\"!"
44      end
45    else
46      render :new
47    end
48  end
```

Quellcode 5.8 Auszug aus `app/controller/sources_controller.rb`

5.2.5 Accountverwaltung

Die Accountverwaltung unterscheidet zwischen internen bzw. lokalen Accounts, welche per E-Mail mittels einer Einladung angelegt werden können, und Accounts, welche sich über den SSO-Dienst Shibboleth der Universität Bremen authentifizieren. Die Eingabemasken zum Anlegen neuer Benutzer*innen können der Abbildung 5.6 *Accountverwaltung* entnommen werden.

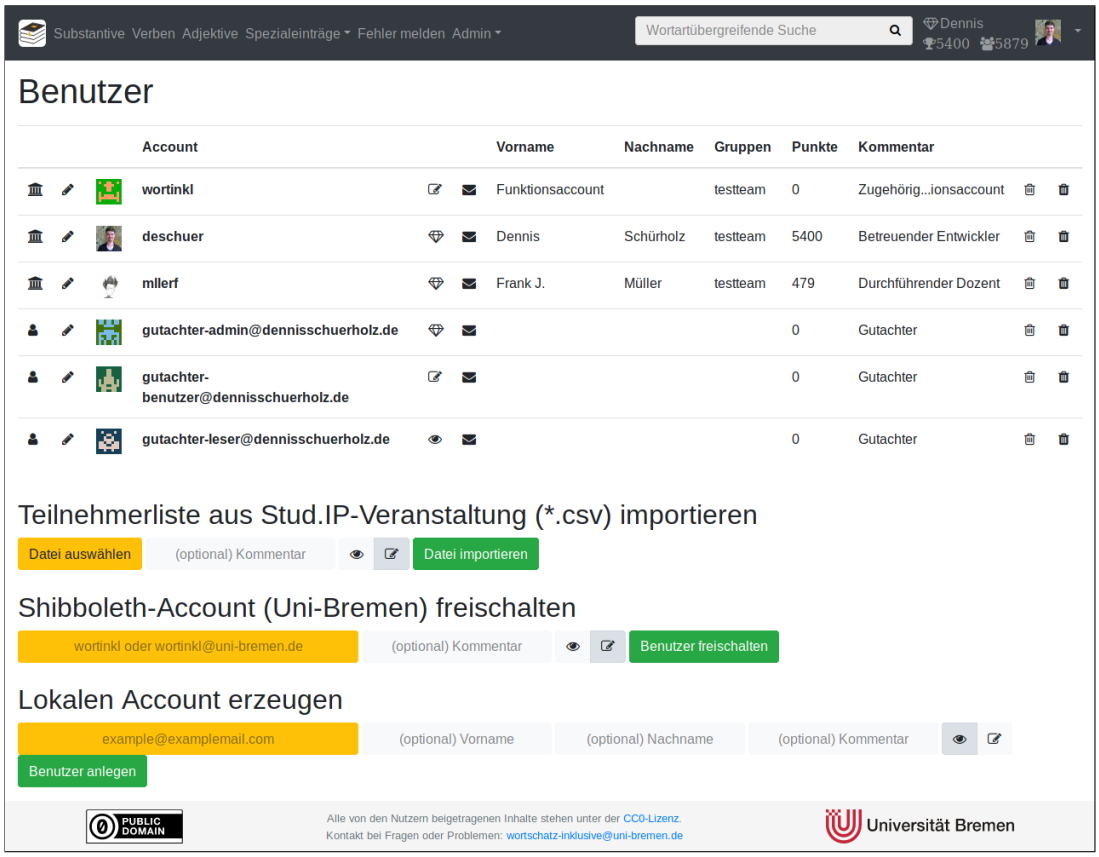

Abbildung 5.6 Accountverwaltung

Für lokale Accounts per E-Mail wird auf die Funktionalität des Gems *devise_invitable*[12] zurückgegriffen, welches erlaubt Einladungsmails zu versenden und damit den Benutzer*innen ebenfalls ermöglicht mit Bestätigung der Einladung ein eigenes Passwort für den weiteren Zugriff auf das System zu setzen. Auf der Seite der Shibboleth-Accounts wird lediglich der universitäre Accountname zur Nutzung des Systems »freigeschaltet«, hierbei kann auch eine ganze Liste von Accounts (in Form einer Teilnehmerliste aus einer Stud.IP-Veranstaltung) in das System übernommen werden. Die Vorgehensweise beim Import von Benutzer*innen kann dem Quellcode 5.9 *Auszug aus* `app/models/user.rb` ab Zeile 43 entnommen werden. Es wird hierbei optimistisch angenommen, dass es sich um Stud.IP-Exporte handelt und diese

[12] *https://rubygems.org/gems/devise_invitable*, abgerufen am: 17.07.2019

mit UTF-8-Kodierung vorliegen (ab Stud.IP 4.0 immer der Fall). Zur Nutzung von Shibboleth wurde beim Zentrum für Netze eine entsprechende Freischaltung beantragt und ein Zertifikat hinterlegt. Auf der Ebene des Softwaresystems wird das Gem *omniauth-shibboleth*[13] eingesetzt und das Apache-Modul `shib2`[14] (siehe Abschnitt 5.1 *Systementwicklungsumgebung*) eingebunden. Da beim Anmelden über Shibboleth Vor- und Nachname sowie die E-Mailadresse aus den hinterlegten Stammdaten bei der Universität übernommen werden (siehe Zeile 27 bis 29 in Quellcode 5.9 *Auszug aus* ***app/models/user.rb***), ist diese Angabe beim Freischalten über die Accountverwaltung optional.

```
24  def self.from_omniauth(auth)
25    user = find_by_provider_and_uid(auth.provider, auth.uid)
26    if user
27      user.given_name = auth.info.given_name if user.given_name.blank?
28      user.family_name = auth.info.family_name if user.family_name.blank?
29      user.email = auth.info.email if user.email.blank?
30      user.save
31    end
32    user
33  end
      ⋮
43  def self.import(file, comment=nil)
      ⋮
50    CSV.foreach(file.path, headers: true, col_sep: ';', encoding: 'UTF-8') do |↩
        row|
51      if User.exists?(uid: row[5])
52        u = User.find_by(uid: row[5])
53        unless u.is_admin?
54          u.comment = comment
55          u.save
56          updated = updated + 1
57        end
58        next
59      end
60
61      hash = Hash.new
62      hash[:uid] = row[5]
63      hash[:email] = "#{row[5]}@uni-bremen.de"
64      hash[:given_name] = row[2]
65      hash[:family_name] = row[3]
66      hash[:provider] = 'shibboleth'
67      hash[:points] = 0
68      hash[:comment] = comment
```

[13] *https://rubygems.org/gems/omniauth-shibboleth*, abgerufen am: 17.07.2019
[14] *https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPApacheConfig*, abgerufen am: 04.06.2019

```
69      User.create! hash
70      count = count + 1
71    end
```

Quellcode 5.9 Auszug aus `app/models/user.rb`

Benutzer*innen jeder Art können Schreibrechte sowie administrative Rechte zugewiesen werden. Mit administrativen Rechten stehen den Benutzer*innen weitere Funktionen wie der Zugriff auf die Account- und Quellenverwaltung (vgl. Abschnitt 5.2.4 *Quellenimport*) und die Möglichkeit zum Massenediteren und Löschen von Datensätzen (vgl. Abschnitt 5.2.2 *Listenansicht*) zur Verfügung.

5.2.6 Programmschnittstellen/API

Die Programmierschnittstelle bzw. API des Systems wird durch die Rails eigenen Konventionen größtenteils vorgegeben. Dabei wird zu jeder Route der grafischen Benutzungsoberfläche automatisch eine Entsprechung im JSON-Format erzeugt (bspw. `/nouns` und `/nouns.json`). Ein lesender Zugriff findet dabei über einen GET-Request auf die entsprechenden Routen (bspw. `/nouns/all/short.json`) statt. Eine Übersicht häufiger Routen kann der Tabelle 5.1 *Übersicht häufig verwendeter API-Routen* entnommen werden. In dieser Tabelle wird mit `NCon` die Klasse `NounsController` (vgl. Quellcode A.4 *app/controller/nouns_controller.rb*) und mit `WCon` die Klasse `WordsController` (vgl. Quellcode A.10 *app/controller/words_controller.rb*) abgekürzt dargestellt.

Methode	Pfad/Route	Controller#Action	Beschreibung
GET	`/nouns.json`	`NCon#`*index*	Liste aller Substantive mit Pagination
GET	`/nouns/all.json`	`NCon#`*index_all*	Liste aller Substantive ohne Paginati-on
GET	`/nouns/all/short.json`	`NCon#`*index_all_short*	Liste aller Substantive ohne Pagina-tion in Kurzform (nur *id*s und *name* (Grundform))
GET	`/nouns/<id>.json`	`NCon#`*show*	Alle zum Substantiv hinterlegten Da-ten (JSON-Objekt)
POST	`/nouns.json`	`NCon#`*create*	Erzeugt ein neues Substantiv anhand des gesendeten JSON-Objektes bzw. der POST-Parameter
PATCH/ PUT	`/nouns/<id>.json`	`NCon#`*update*	Passt das Substantiv anhand des ge-sendeten JSON-Objektes bzw. der POST-Parameter an
DELETE	`/nouns/<id>.json`	`NCon#`*destroy*	Löscht das Substantiv
POST	`/nouns/bulk_update`	`NCon#`*bulk_update*	Führt eine Änderung an mehre-ren Datensätzen gemäß der POST-Parameter durch
GET	`/words/all/short.json`	`WCon#`*index_all_short*	Liste aller Wörter ohne Pagination in Kurzform

Tabelle 5.1 Übersicht häufig verwendeter API-Routen

Im Gegensatz zu den Routen aus Tabelle 5.1, welche JSON und HTML als Formate bearbeiten, können die Routen aus Tabelle 5.2 nur im HTML-Format bedient werden, da diese Formulare darstellen, um ein Bearbeiten bzw. Erzeugen über die grafische Benutzungsoberfläche zu ermöglichen. Sie finden hier Erwähnung als Kontrast zu den API-Funktionalitäten, da sie nicht zu den eigentlichen Programmier-schnittstellen gehören.

Methode	Pfad/Route	Controller#Action	Beschreibung
GET	`/nouns/<id>/edit`	`NCon#`*edit*	Formular zum Bearbeiten des Substantivs
GET	`/nouns/new`	`NCon#`*new*	Formular zum Erzeugen eines Substantivs
GET	`/words/<id>/edit`	`WCon#`*edit_redirect*	Weiterleitung auf das Bearbeitungsformular des Wortes (je nach Wortart)

Tabelle 5.2 Übersicht häufig verwendeter Routen mit HTML Ausgabe ohne API-Entsprechung

Alle *index**-Routen können auch über die in Abschnitt 5.2.2 *Listenansicht* eingeführte Suchfunktion gefiltert und sortiert werden. Hierfür können die Suchschlüssel entsprechend als URL-Parameter im GET-Request angegeben werden, sie werden jeweils als Wertepaar `filterrific[<search_key>]=<value>` angegeben. Beispiele für die Subschlüssel in Anlehnung an den Auszug in Quellcode 5.1 sind:

`search_key`	`value`	Beschreibung
`[search_wordstarts]`	`<string>`	Filtert nach Wörtern mit der Zeichenkette zu Beginn der Grundform
`[with_topic]`	`<topic_id>`	Filtert nach Wörtern zum mittels *id* referenzierten Thema

`[with_keywords][conjunction]`
`[with_keywords][keywords][]`
`[with_keywords][keywords][]`
	`[AND|OR]`
	`<keyword_id>`
	`<second_keyword_id>`
		Filtert nach Wörtern mit den referenzierten Stichwörtern

`[is_foreign]`	`[CHECKED|UNCHECKED|` `NONE|EMPTY]`	Filtert nach Fremdwörtern

Tabelle 5.3 Übersicht einiger Suchschlüssel

Die JSON-Daten für die Rückgabe werden mittels *jbuilder*[15] erzeugt und enthalten je nach verwendeten API-Endpunkt (vgl. Tabelle 5.1) mehr oder weniger der Attribute der spezifischen Wortart. Die Antwort auf die Anfrage `GET /words/all/short.json` wird mit dem Quellcode 5.10 *app/views/words/index.short.json.jbuilder* erzeugt und enthält in den Elementen des erzeugten JSON-Array neben *id* und *name* (Grundform) auch die Wortart, welcher das jeweilige Wort angehört, und die *id* innerhalb der dazugehörigen Tabelle. Dadurch lässt sich eine Detailanfrage an bspw. die entsprechende »show«-Route (vgl. Tabelle 5.1) der Wortart formulieren. Bei der Anfrage der entsprechenden Route beim Substantiv (`GET /nouns/all/short.json`, siehe Quellcode 5.11 *app/views/nouns/index.short.json.jbuilder*) fällt in der Antwort die (bereits bekannte) Wortart weg und die *actable_id* wird stattdessen als *noun_id* bezeichnet. Die verbleibenen Wortarten werden äquivalent gebildet.

[15] *https://rubygems.org/gems/jbuilder*, abgerufen am: 17.07.2019

```
1 json.words @words do |word|
2   json.id word.id
3   json.name word.meaning.empty? ? word.name : "#{word.name} [#{word.meaning}]"
4   json.actable_type word.actable_type
5   json.actable_id word.actable_id
6 end
```

Quellcode 5.10 `app/views/words/index.short.json.jbuilder`

```
1 json.nouns @nouns do |noun|
2   json.id noun.acting_as.id
3   json.noun_id noun.id
4   json.name noun.meaning.empty? ? noun.name : "#{noun.name} [#{noun.meaning}]"
5 end
```

Quellcode 5.11 `app/views/nouns/index.short.json.jbuilder`

Wie den Quellcode-Auszügen in Quellcode 5.5 *Auszug aus* ***app/views/nouns/_form.html.haml*** und Quellcode 5.6 *Auszug aus* ***app/assets/javascripts/initializers.coffee*** bereits entnommen werden konnte, nutzt die grafische Benutzungsoberfläche bereits diese Teile der API um Auswahllisten mittels JavaScript nach dem Seitenaufbau nachzuladen. Dies wird bspw. auch von den Filtern aus Abschnitt 5.2.2 verwendet.

In den bislang beschriebenen Kontexten kommt die API zum Einsatz, während eine Benutzer*in an der grafischen Benutzngsschnittstelle angemeldet ist. Dadurch findet die Authentifizierung und Autorisierung über die Sitzungsverwaltung von *devise*[16] statt. Damit die API auch unabhängig von der Verwendung im Browser (z. B. in einer App oder einfach konsolenbasiert per `curl` o. Ä.) nutzbar ist, sind parallel für den Zugriff auf JSON-Endpunkte über das Gem *simple_token_authentication*[17] eine Authentifizierung mittels Zugriffstoken möglich. Dabei verfügt jede Benutzer*in über einen persönlichen Zugriffstoken, welcher über die Profilseite (vgl. Abbildung 5.1 *Startseite für angemeldete Benutzer*innen (eigenes Profil)*) abrufbar ist.

In einer Anfrage an die API kann damit die E-Mailadresse und der Zugriffstoken als HTTP-Header (`X-User-Email` und `X-User-Tocken`) oder als URL-Parameter (`user_email` und `user_token`) übergeben werden. Beispiele für den Aufruf sind dann:

```
curl https://wortschatz.dennisschuerholz.de/words/all/short.json \
  -H "X-User-Email: gutachter-leser@dennisschuerholz.de" \
  -H "X-User-Token: cCLuyhKsGG1FjZbnTexJ"
curl https://wortschatz.dennisschuerholz.de/words/all/short.json\?\
  user_email\=gutachter-leser@dennisschuerholz.de\&\
  user_token\=cCLuyhKsGG1FjZbnTexJ
```

[16] *https://rubygems.org/gems/devise*, abgerufen am: 17.07.2019
[17] *https://rubygems.org/gems/simple_token_authentication*, abgerufen am: 17.07.2019

5.3 Usabilitytest

Durch den frühzeitigen produktiven Einsatz in Seminaren von Prof. Dr. Frank J. Müller kann das Feedback von Nutzer*innen schnell erfasst werden und so bereits in der frühen Entwicklungsphase berücksichtigt werden. In diesem Prozess durchläuft die grafische Benutzungsoberfläche sowohl für die Übersicht und das Filtern (vgl. Abschnitt 5.2.2 *Listenansicht*) als auch für die Bearbeitungsansicht (vgl. Abschnitt 5.2.3 *Bearbeitungsansicht*) mehrere Iterationen. Hierbei sind schon durch die natürliche Nutzung die häufigen Anwendungsfälle gemäß Abschnitt 4.2 *Anwendungsfälle* durch Prof. Dr. Frank J. Müller, seine Mitarbeiter*innen und Studierenden einer fortlaufenden Prüfung unterzogen. Mit diesem Vorgehen werden Rückmeldungen gezielt eingearbeitet und somit der Gestaltungsprozess der grafischen Benutzungsoberfläche agil durchgeführt. Zusatzfunktionen (wie bspw. bestimmte Filter) werden dabei nach konkretem Bedarf zum Projektumfang ergänzt und durch die Nachfrage bestimmt priorisiert umgesetzt. Auf eine formale Nutzer*innenstudie außerhalb des bereits erfolgten produktiven Einsatzes des Systems wird im Rahmen dieser Bachelorarbeit verzichtet.

Kapitel 6

Ausblick

In der Bearbeitung des Projektes (siehe Abschnitt 3.1) bzw. dem daraus resultierenden Ergebnis in Form der implementierten Webanwendung bleiben Punkte aus der initialen Projektbeschreibung eine Implentation schuldig bzw. stehen diese weiteren Diskussionen gegenüber offen. Darüber hinaus gibt es ebenfalls weitere Aspekte und auch Anknüpfungnspunkte aus dem Gesamtkontext des Themengebietes, welche in eigenständigen Projekten im Anschluss an diese Arbeit aufgegriffen werden können. Im folgenden wird auf einige dieser Punkte mit Ideen bzw. Anregungen zur weiteren Arbeit eingegangen.

Zunächst sind Teile der in dem Projektvorhaben (siehe Abschnitt 3.1) und den Projektanforderungen (siehe Abschnitt 4.1) im Rahmen dieser Arbeit noch nicht umgesetzt. Diese werden in weiterer Zusammenarbeit mit Prof. Dr. Frank J. Müller zur Umsetzung seiner Projektideen bzw. des Gesamtvorhabens im Anschluss an die Arbeit umgesetzt. Darunter fällt konkret auch die Umsetzung der Verwaltung bzw. Zuordnung von Audio- und Bilddateien zu den erfassten Datensätzen.

Außerdem ist die Publizierung der Plattform (mit öffentlicher lesender Zugriffsmöglichkeit) und die ebenfalls geplante Veröffentlichung des Quellcodes unter CC-BY (siehe Abschnitt 2.1.2) noch ausstehend. Ein weiterer Ansatz, welcher evaluiert und ggf. weitergehend bearbeitet werden kann, umfasst ein mögliches Review-System um Änderungen an der Datenbasis von beliebigen Nutzer*innen zu erlauben und gleichzeitig die Qualität der Daten gewährleisten zu können.

Mögliche Anknüpfungspunkte für zukünftige Projekte könnten sein:

- Die Entwicklung einer App mit Anknüpfung an die Programmierschnittstelle des Systems (siehe Abschnitt 5.2.6), um damit bspw. Kinder zum selbstständigen Lernen anzuleiten oder als ein mobiles Nachschlagewerk zu fungieren.
- Die Verbindung oder eigenständige Entwicklung eines Materialienmarktplatzes um die einfache Möglichkeit zu schaffen, aufbereitete Lehr- und Lernmaterialien in eine Datenbank zurückzuspiegeln um einen Austausch im Gedanken der OER zu ermöglichen. Da eine der Hauptaufgaben des entwickelten Systems die Grundlagensammlung zum Erzeugen solcher Materialien darstellt, ist die Sammlung und Bereitstellung dieser Materialien ein logischer Ausbauschritt.

- Vor allem im Bereich der Themen- und Quellenkategorisierung bspw. nach Grundwortschätzen verschiedener Schulbezirke bzw. Bundesländer ist es vorstellbar, dass bspw. von Hochschulen, Schulen oder Landesschulinstituten eigene Instanzen der Plattform betrieben werden. Um diese dezentralisierten Plattformen insbesondere im Bereich der grundlegenden (Meta-)Daten nicht jeweils losgelöst zu betrachten ist ein Konzept zur Föderation zu entwickeln. Hierdurch würde es möglich an einer gemeinsamen Datenbasis der Grundlagendaten zu arbeiten und gleichzeitig für lokale Projekte bzw. Lehrpläne zugeschnittene Einstellungen oder Sammlungen anzufertigen.
- Um die Verbreitung des Systems auch über die deutschsprachigen Regionen hinaus zu ermöglichen sollte die Möglichkeit der Mehrsprachigkeit bzw. Internationalisierung evaluiert werden. Neben der Oberflächengestaltung müssen ggf. auch abweichende Datenfelder vorgesehen werden, da die gespeicherten (Meta-)Daten selbst den Gegebenheiten der deutschen Grammatik unterliegen.

Kapitel 7

Fazit

Diese Bachelorarbeit bringt ein System als Ergebnis hevor, welches bereits durch Prof. Dr. Frank J. Müller produktiv eingesetzt wird. Im Rahmen dieses Kontextes ist das System auch über die Bachelorarbeit hinaus bekannt und wird in überregionalen Forschungs- und Unterstützungsprojekten eingesetzt bzw. es bestehen Anfragen für eben diese Nutzung und Kooperation. Dieses Ergebnis ist durch die Enge Zusammenarbeit in Analyse der Anforderungen und der Spezifikation des Softwaresystems mit Prof. Dr. Frank J. Müller entstanden und spiegelt damit einen realen Softwareentwicklungsprozess mit einem Kunden bzw. Projektpartner wieder. Der im Rahmen dieser Bachelorarbeit entwickelte Systementwurf (vgl. Kapitel 4) spiegelt sich in dem real umgesetzten System (vgl. Kapitel 5) gut wieder und entspricht den erarbeiteten Anforderungen.

Um den Umfang des Projektes für den Rahmen der Bachelorarbeit kleiner zu halten wurden jedoch auch nicht alle angedachten Datenfelder (vgl. Abschnitt A.9.1 *Datenbankstruktur, Initialversion*) umgesetzt. In diesem Punkt wird für die vollständige Projektumsetzung außerhalb der Bachelorarbeit noch weitere Entwicklungszeit investiert werden müssen. Der Systementwurf (vor allem im Bereich der grafischen Benutzungsoberfläche) wurde mit dem Projektpartner iterativ erweitert. Dabei ist festgestellt worden, dass eine testgetriebene Entwicklung in einigen Bereichen für eine höhere Kontinuität im Entwicklungsprozess gesorgt hätte. Auf dieser Basis hätten wiederkehrende Fehler vermieden bzw. schneller erkannt werden können, in der weiteren Arbeit am Projekt muss dies aufgegriffen werden. Eine ausführliche Evaluation mit Endnutzer*innen fand im Rahmen der Bachelorarbeit nicht statt (vgl. Abschnitt 5.3 *Usabilitytest*), dies sollte ebenfalls im Vorfeld einer weiteren Arbeit am System mit aufgegriffen und umgesetzt werden.

Das Projekt ist durch Verwendung von verbreiteten Frameworks (bspw. *rails*[1] und *devise*[2]) leicht erweiterbar und anpassbar. Hiervon konnte bereits während der Implementierung profitiert werden, auch bei der zukünftigen Weiternutzung und damit verbundenen Weiterentwicklung wird dies von großem Vorteil sein.

[1] *https://rubygems.org/gems/rails*, abgerufen am: 17.07.2019
[2] *https://rubygems.org/gems/devise*, abgerufen am: 17.07.2019

Anhang A

Appendix

A.1 Abbildungsverzeichnis

A.2 Tabellenverzeichnis

A.3 Quellcodeverzeichnis

A.4 Literaturverzeichnis

Bootstrap-Team (2019a). *About*. URL: *https://getbootstrap.com/docs/4.3/about/overview/* (abgerufen am 15. 04. 2019).

– (2019b). *Grid system*. URL: *https://getbootstrap.com/docs/4.3/layout/grid/* (abgerufen am 20. 06. 2019).

Brause, Martin und Manfred Schulz (21. Juni 2017). »Open Educational Resources in der Schule – Bildung in der digitalen Welt«. In: *Synergie. Fachmagazin für Digitalisierung in der Lehre* 03.

Buschmann, Frank, Regine Meunier, Hans Rohnert, Peter Sommerlad und Michael Stal (1996). *Pattern-oriented software architecture. A system of patterns*. Repr. Chichester [u.a.]: Wiley. ISBN: 0471958697. URL: *https://suche.suub.uni-bremen.de/peid=B19446563*.

Diepenmaat, Joost und Remco van 't Veer (28. Juni 2005). *The Model View Controller pattern in web applications*. Zeekat Software Development. URL: *https://zeekat.nl/articles/mvc-for-the-web.html* (abgerufen am 25. 03. 2019).

e-teaching.org (10. Dez. 2018). *Open Educational Resources*. URL: *https://www.e-teaching.org/didaktik/recherche/oer* (abgerufen am 21. 06. 2019).

Kück, Dietmar (21. Juni 2017). »OER in die Schule! Freie Bildungsmaterialien für Unterricht und modernes Lernen nutzen«. In: *Synergie. Fachmagazin für Digitalisierung in der Lehre* 03.

Morsy, Hussein und Tanja Otto (2012). *Ruby on Rails 3.1. Das Entwickler-Handbuch*. 2., aktualisierte Aufl. Galileo computing. Bonn: Galileo Press. ISBN: 3836214903. URL: *https://suche.suub.uni-bremen.de/peid=B61834505*.

Müller, Frank J. (2016). »Inklusive Open Educational Resources. Wie frei verfügbare Bildungsmaterialien im Umgang mit Heterogenität helfen können«. In: *Schweizerische Zeitschrift für Heilpädagogik* 22 4. ISSN: 1420-1607. URL: *http://nbn-resolving.org/urn:nbn:de:0111-pedocs-155864*.

– (2019a). *Chancen und Herausforderungen staatlich finanzierter, frei verfügbarer Bildungsmaterialien (OER) am Beispiel der Plattform ndla.no in Norwegen. Ein Weg zu mehr Inklusion?* München: ZLL21 e.V. ISBN: 978-3-9818942-4-0. URL: *http://nbn-resolving.org/urn:nbn:de:0111-pedocs-169937*.

– (2019b). *#UnsereWörter. inklusive freie Bildungsmaterialien zum interessengeleiteten Schriftspracherwerb - ein Mitmachprojekt*. URL: *https://wortschatzinklusive.uni-bremen.de/* (abgerufen am 19. 07. 2019).

Muuß-Merholz, Jöran (13. Juli 2017). *Creative_Commons_Lizenzspektrum_DE*. URL: *https://commons.wikimedia.org/wiki/File:Creative_Commons_Lizenzspektrum_DE.svg* (abgerufen am 12. 04. 2019). Lizenziert unter Creative Commons BY-SA.

Steffens, Yannic, Inga Lotta Schmitt und Sandra Assmann (24. Mai 2018). »ZuhOERen. Das BMBF-Projekt You(r) Study: Studieren zwischen Eigensinn und Unbestimmtheit«. In: *Synergie. Fachmagazin für Digitalisierung in der Lehre* 05.

UNESCO (2017). *Open Educational Resources*. UNESCO. URL: *https://www.unesco.de/bildung/open-educational-resources* (abgerufen am 18. 03. 2019).

Wikibooks (2018). *Computer Science Design Patterns — Wikibooks, The Free Textbook Project*. URL: *https://en.wikibooks.org/w/index.php?title=Computer_Science_Design_Patterns&oldid=3447180* (abgerufen am 22. 03. 2019).

A.5 Abkürzungsverzeichnis

API Application Programming Interface, S. 17, 19, 20, 48, 49, 51, 62, *Glossary:* Application Programming Interface

CC Creative Commons, S. 4, 5

MVC Model-View-Controller, S. 3, 6–8, 26

OER Open Educational Ressources, S. 1–6, 12–14, 17, 53, 62, *Glossary:* Open Educational Ressources

Rails Ruby on Rails, S. 8, 9, 26, 31, 32, 48, 62, 63, *Glossary:* Ruby on Rails

SSO Single-Sign-On, S. 16, 46, 62, 65, *Glossary:* Single-Sign-On

A.6 Glossar

Application Programming Interface

Eine API ist eine Schnittstelle, welche für die Anbindung von Drittsystemen konzipiert ist. Über diese werden bspw. Daten in maschinenlesbaren Formaten ausgetauscht. Im Gegensatz dazu existieren grafische Benutzungsoberflächen für die Interaktion mit Endnnutzer*innen. (vgl. Abschnitt 5.2.6 *Programmschnittstellen/API*)

S. 17, 62

Controller

Nimmt Anforderungen durch Interaktion mit den Benutzer*innen von der View entgegen und verwaltet die Daten des Models. Dadurch beinhaltet der Controller die einzigen Klassen, welche eine höhere Geschäftslogik aufweisen und bspw. Manipulationen an den Daten vornehmen.

S. 6–9, 26, 63, 65

CSS

Cascading Style Sheets (CSS) kommt vor allem in Kombination mit HTML zum Einsatz und bietet die Möglichkeit der visuellen Gestaltung der Inhalte.

S. 7, 9

CSV

Das Datenformat CSV (comma-separated values) legt den strukturierten Aufbau einer Textdatei fest wonach Datensätze i. d. R. durch Zeilenumbrüche und Datenfelder innerhalb dieser Datensätze mit einem Trennzeichen wie einem Komma oder Semikolon getrennt werden. Um innerhalb der Datenfelder auch das jeweilige Trennzeichen nutzbar zu machen ist auch die Verwenung von Anführungszeichen als Feldbegrenzer (innerhalb derer das Trennzeichen als regulärer Dateninhalt behandelt wird) möglich.

Dateien in diesem Datenformat tragen die Endung `.csv` und werden Beispielsweise von Tabellenkalkulationsprogrammen und zum Datenimport und Datenexport von verschiedenen Softwaresystemen verwendet.

S. 12, 43, 64

Full-Stack Framework

Ein Full-Stack Framework soll einen schnellen Entwicklungsprozess ermöglichen, indem es die Entwicklungskomponenten von der grafischen Benutzungsoberfläche bis zum Datenbanksystem vereint. Ein solches Framework ist bspw. Ruby on Rails.

S. 8, 64

Gamification

Der Einsatz spieltypischer Elemente (bspw. Punkte, Ranglisten, Abzeichen) in spielfremden Kontexten wie einer Lernumgebung wird als Gamification bezeichnet.

S. 13, 18

Genus
Beschreibt das grammatische Geschlecht eines Wortes. Im deutschen wird unterschieden zwischen Maskulinum (z. B. *der Baum*), Femininum (z. B. *die Schule*) und Neutrum (z. B. *das Kind*).
S. 43

Haml
Haml ist eine vereinfachte Auszeichnungssprache aus welcher z. B. HTML-Code erzeugt werden kann. Sie zeichnet sich dadurch aus, dass sie eine sehr kompakte Syntax hat.
S. 7

HTML
HTML wird als Auszeichnungssprache für Webdokumente verwendet. Es gliedert die Dokumente auf semantischer Ebene.
S. 7, 8, 49, 50, 58, 63, 64

JavaScript
JavaScript ist eine Skriptsprache. Sie wird eingesetzt um Interaktionen von Benutzer*innen mit Webseiten ohne Neuladen dieser durchzuführen. Es bietet bspw. auch die Möglichkeit, zusätzliche Inhalte dynamisch vom Server nachzuladen.
S. 7, 40, 51

JSON
Das Datenformat JSON (JavaScript Object Notation) ist zum Austausch von Datensätzen zwischen verschiedenen Anwendungen (ähnlich zu CSV oder XML) entwickelt worden. JSON-Dokumente sind dabei auch JavaScript interpretierbar und somit direkt als Datenstrukturen nutzbar, es existieren Interpreter für alle verbreiteten Sprachen inklusive des im Kontext der Arbeit verwendeten Ruby.
S. 8, 48–51

Model
Dient der Repräsentation der Daten, diese enthalten keine weitergehende Geschäftslogik und werden lediglich vom View zur Darstellung verwendet. Es sind in Objektorientierten Programmiersprachen auch Vererbungen innerhalb der Modelle möglich.
S. 6–9, 26, 63, 65

Open Educational Ressources
»Open Educational Ressources sind Bildungsmaterialien jeglicher Art und in jedem Medium, die unter einer offenen Lizenz stehen.« (UNESCO 2017, vgl. Abschnitt 2.1.1 *Definition von OER*)
S. 1, 3, 4, 62

Ruby on Rails
Ruby on Rails, bzw. kurz Rails, ist ein Full-Stack Framework für die Programmiersprache Ruby und ist ebenfalls in dieser geschrieben. (vgl. Abschnitt 2.3.1 *Ruby on Rails*)
S. 8, 26, 31, 62, 63

Shibboleth

Shibboleth ist ein Dienst zur Authentifizierung und Autorisierung (vgl. Single-Sign-On) von Webanwendungen. Die Universität Bremen tritt bei Shibboleth als Identity Provider auf[1].

S. 16, 17, 32, 46, 47, 65

Single-Sign-On

Single-Sign-On beschreibt, dass eine Benutzer*in sich nur einmalig Authentifizieren muss um auf verschiedene Dienste zugreifen zu können für die eine Autorisierung besteht. Ein Beispiel für einen SSO-Dienst ist Shibboleth.

S. 16, 62, 65

View

Stellt die Daten des Models dar und ermöglicht die Interaktion mit den Benutzer*innen. Eine Interaktion wird dem Controller mitgeteilt, sodass die Anzeige ohne Geschäftslogik auskommt. Die Anzeige ist dabei häufig eine Komposition aus mehreren Teilkomponenten. In der selben Anwendung kann es mehrere Views für verschiedene Aufgaben geben, bspw. zum Anzeigen der Daten in der Übersicht oder einer Detailsicht, sowie zum Bearbeiten.

S. 6–9, 33, 40, 63, 64

[1] *https://onlinetools.zfn.uni-bremen.de/server/content/aai/shibboleth-anwender.php*, abgerufen am: 16.04.2019

A.7 Weitere Abbildungen

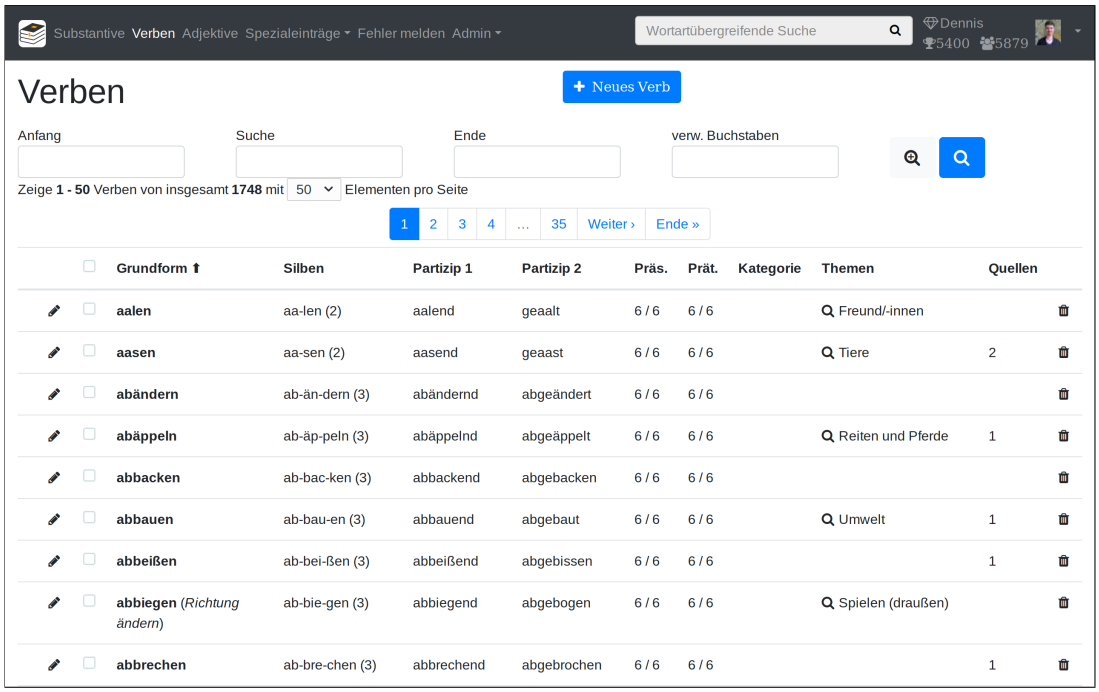

Abbildung A.1 Listenansicht der Verben

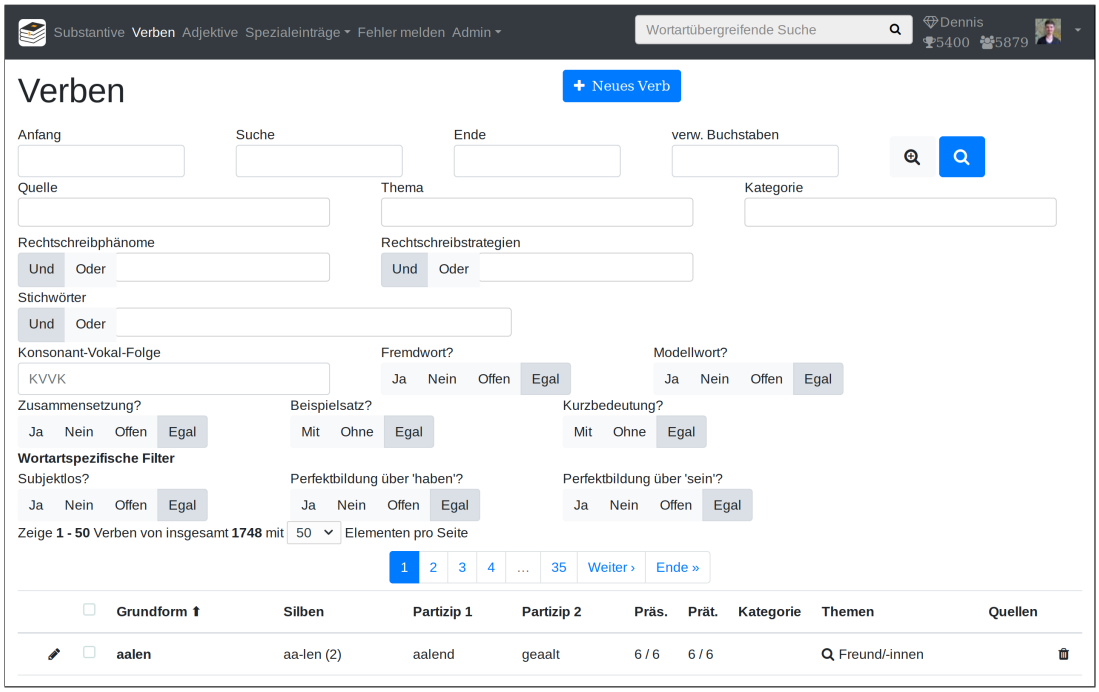

Abbildung A.2 Filteroptionen der Verben

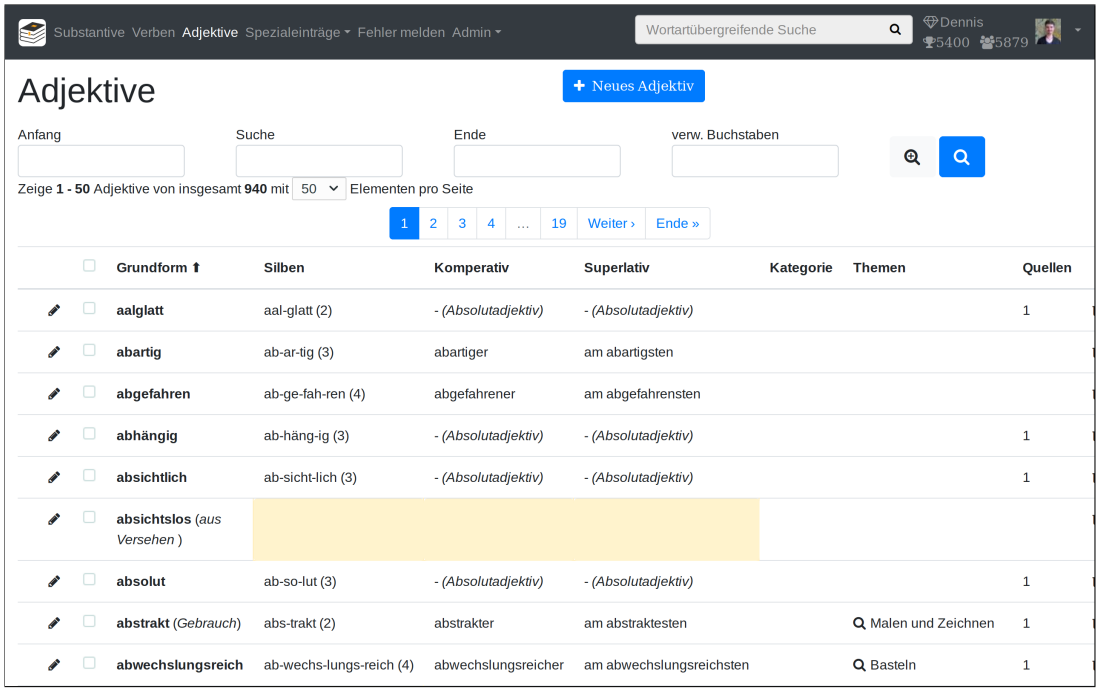

Abbildung A.3 Listenansicht der Adjektive

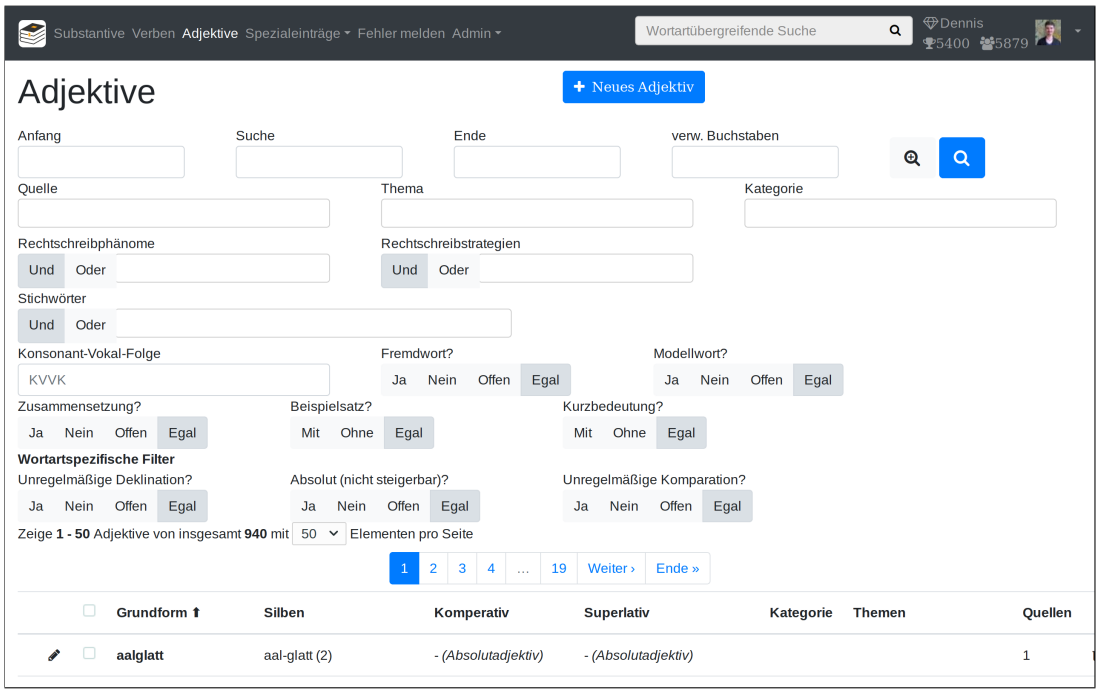

Abbildung A.4 Filteroptionen der Adjektive

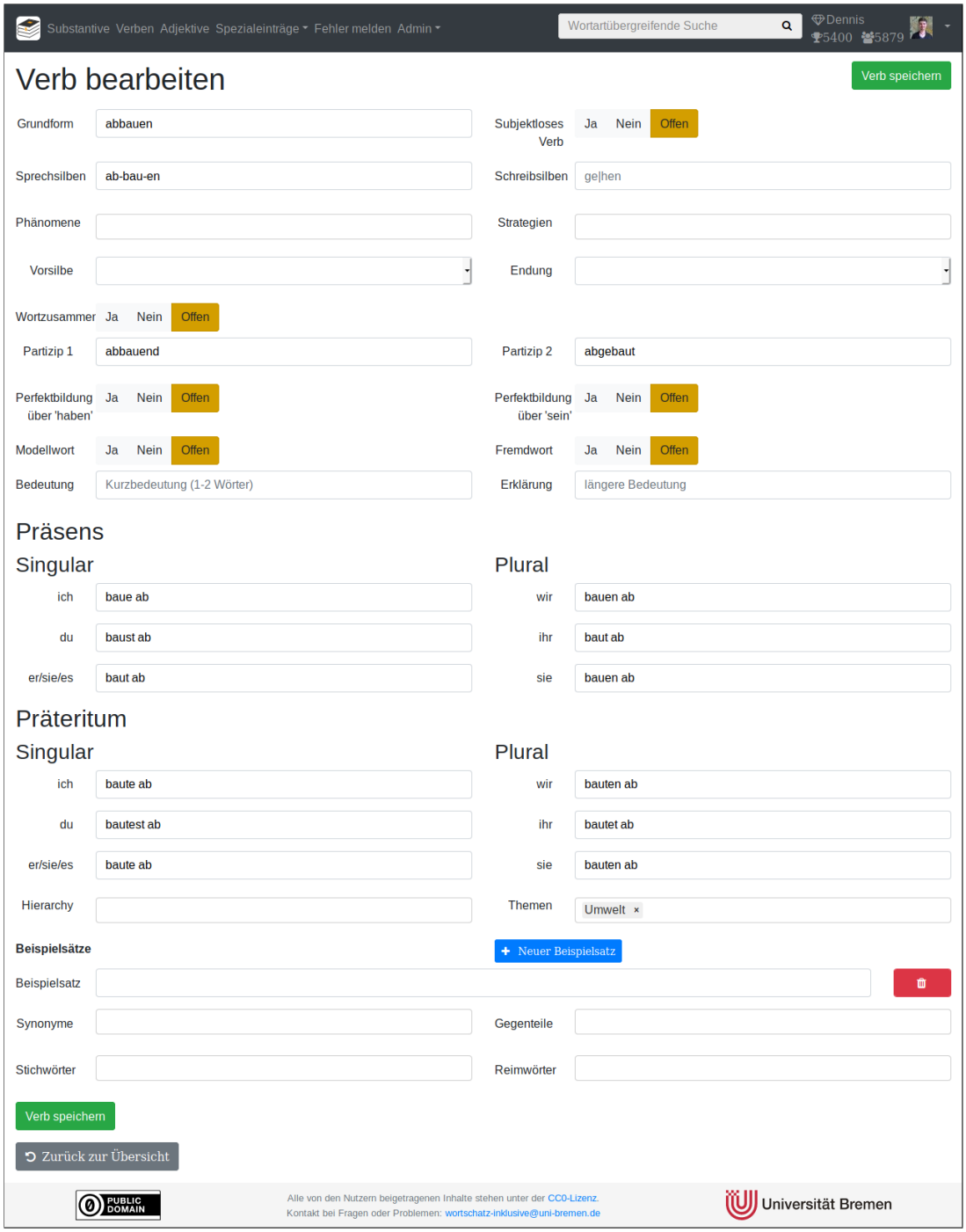

Abbildung A.5 Bearbeitungsansicht der Verben

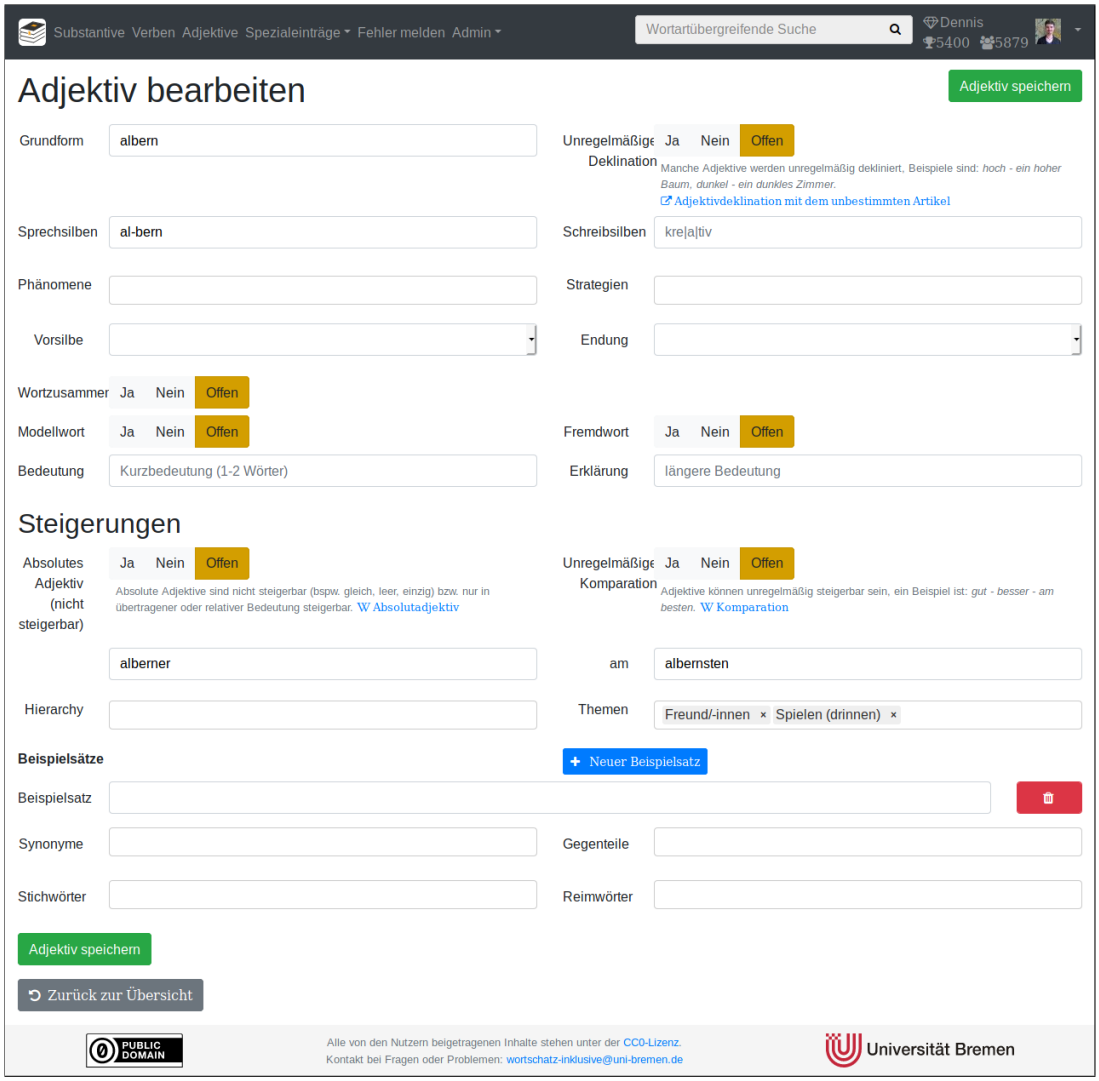

Abbildung A.6 Bearbeitungsansicht der Adjektive

A.8 Weiterer Quellcode

In diesem Abschnitt werden zum Hauptteil ergänzende Quellcode-Abschnitte sowie vollständige Listings von im Hauptteil nur auszugsweise eingebundenen Dateien abgebildet.

Eine vollständige Version der im Rahmen dieser Arbeit umgesetzten Anwendung ist mit Stand der Abgabeversion auf der CD (siehe Abschnitt A.10 *Digitaler Anhang – CD*) zur Arbeit beigefügt. Die weitere Arbeit an der Software kann im Gitlab des Fachbereich 3 unter *https://gitlab.informatik.uni-bremen.de/word/wordapp* auf Einladung bzw. zukünftig auf öffentlichen Plattformen verfolgt werden.

Quellcode A.1 `lib/word_filter.rb`

```
require 'active_support/concern'
module WordFilter
  extend ActiveSupport::Concern

  included do
    filterrific(
      default_filter_params: {
        sorted_by: 'name_asc'
      },
      available_filters: [
        :sorted_by,
        :search_wordstarts,
        :search_wordends,
        :search_wordcontains,
        :search_wordconsists,
        :search_wordquery,
        :with_source,
        :with_topic,
        :with_hierarchy,
        :with_phenomenons,
        :with_strategies,
        :with_keywords,
        :with_genus,
        :with_syllables_count,
        :search_consonant_vowel,
        :has_example_sentence,
        :has_meaning,
        :is_foreign,
        :is_prototype,
        :is_compound,
        :is_singularetantum,
        :is_pluraletantum,
```

```
33          :is_subjectless,
34          :is_perfect_haben,
35          :is_perfect_sein,
36          :is_irregular_declination,
37          :is_absolute,
38          :is_irregular_comparison,
39        ]
40      )
41
42      scope :sorted_by, lambda { |sort_option|
43        direction = (sort_option =~ /desc$/) ? 'DESC' : 'ASC'
44        direction_reverse = (sort_option =~ /desc$/) ? 'ASC' : 'DESC'
45        case sort_option.to_s
46        when /^name_/
47          joins(:word).order("LOWER(`name`) #{direction}")
48        when /^plural_/
49          order("LOWER(`plural`) #{direction}, `singularetantum` #{direction}, `↩
              pluraletantum` #{direction}")
50        when /^syllables_/
51          joins(:word).order("LOWER(`syllables`) #{direction}")
52        when /^casus_/
53          order("`casus_empty` #{direction_reverse}")
54        when /^comparative_/
55          order("LOWER(`comparative`) #{direction}, `absolute` #{direction}")
56        when /^superlative_/
57          order("LOWER(`superlative`) #{direction}, `absolute` #{direction}")
58        when /^participle_/
59          order("LOWER(`participle`) #{direction}")
60        when /^past_participle_/
61          order("LOWER(`past_participle`) #{direction}")
62        when /^present_/
63          order("`present_empty` #{direction_reverse}")
64        when /^past_/
65          order("`past_empty` #{direction_reverse}")
66        when /^separable_/
67          order("`separable` #{direction}")
68        when /^inseparable_/
69          order("`inseparable` #{direction}")
70        end
71      }
72
73      scope :search_wordstarts, lambda { |search|
74        return nil if search.blank?
```

```
75        search_wordquery(search + '%')
76      }
77
78      scope :search_wordends, lambda { |search|
79        return nil if search.blank?
80        search_wordquery('%' + search)
81      }
82
83      scope :search_wordcontains, lambda { |search|
84        return nil if search.blank?
85        search_wordquery('%' + search + '%')
86      }
87
88      scope :search_wordconsists, lambda { |search|
89        return nil if search.blank?
90        term = search.downcase.gsub /[^a-zäöüß]/u, ''
91        logger.info "Term: #{term}"
92        joins(:word).where("LOWER(`name`) RLIKE REPEAT('[#{term}]', CHAR_LENGTH(`name`)) COLLATE utf8_bin")
93      }
94
95      scope :search_wordquery, lambda { |search|
96        return nil if search.blank?
97        term = search.downcase.gsub('*', '%').gsub('?', '_')
98        joins(:word).where("LOWER(`name`) LIKE ? COLLATE utf8_bin", term)
99      }
100
101     scope :with_source, lambda { |source|
102         joins(:word).joins("INNER JOIN `sources_words` ON `words`.`id` = `sources_words`.`word_id`").where(sources_words: { source_id: source})
103     }
104     scope :with_topic, lambda { |topic|
105         joins(:word).joins("INNER JOIN `topics_words` ON `words`.`id` = `topics_words`.`word_id`").where(topics_words: { topic_id: topic})
106     }
107     scope :with_hierarchy, lambda { |hierarchy|
108         where(hierarchy: hierarchy)
109     }
110     scope :with_phenomenons, lambda { |phenomenons|
111       return nil if phenomenons.phenomenons.nil?
112       phenomenons.phenomenons.select!{|w| /^[0-9]+$/.match? w}
113       return if phenomenons.phenomenons.empty?
```

```
114       count = ""
115       count = "COUNT(*) = #{phenomenons.phenomenons.length}" if phenomenons.↩
              conjunction == "AND"
116       joins(:word).joins("INNER JOIN `phenomenons_words` ON `words`.`id` = `↩
              phenomenons_words`.`word_id`").where(phenomenons_words: { ↩
              phenomenon_id: phenomenons.phenomenons }).group('`phenomenons_words↩
              `.`word_id`').having(count).distinct
117     }
118     scope :with_strategies, lambda { |strategies|
119       return if strategies.strategies.nil?
120       strategies.strategies.select!{|w| /^[0-9]+$/.match? w}
121       return nil if strategies.strategies.empty?
122       count = ""
123       count = "COUNT(*) = #{strategies.strategies.length}" if strategies.↩
              conjunction == "AND"
124       joins(:word).joins("INNER JOIN `strategies_words` ON `words`.`id` = `↩
              strategies_words`.`word_id`").where(strategies_words: { strategy_id: ↩
              strategies.strategies }).group('`strategies_words`.`word_id`').having↩
              (count).distinct
125     }
126     scope :with_keywords, lambda { |keywords|
127       return if keywords.keywords.nil?
128       keywords.keywords.select!{|w| /^[0-9]+$/.match? w}
129       return nil if keywords.keywords.empty?
130       count = ""
131       count = "COUNT(*) = #{keywords.keywords.length}" if keywords.conjunction ↩
              == "AND"
132       joins(:word).joins("INNER JOIN `keywords` ON `words`.`id` = `keywords`.`↩
              word_id`").where(keywords: { keyword_id: keywords.keywords }).group('↩
              `keywords`.`word_id`').having(count).distinct
133     }
134     scope :with_genus, lambda { |genus|
135       gid = []
136       case genus
137       when 'MASCULINUM'
138         gid = [0, 3, 4, 6]
139       when 'FEMININUM'
140         gid = [1, 3, 5, 6]
141       when 'NEUTER'
142         gid = [2, 4, 5, 6]
143       end
144       #if genus.exclusive?
145       #  gid = [gid.first]
```

```
146       #end
147       where(genus: gid)
148     }
149     scope :with_syllables_count, lambda { |count|
150       joins(:word).where("(LENGTH(`syllables`) - LENGTH(REPLACE(`syllables`, ↩
              '-', '')) + 1) = ?", count)
151     }
152     scope :search_consonant_vowel, lambda { |cv|
153       cv.upcase!
154       where(consonant_vowel: cv)
155     }
156     scope :has_example_sentence, lambda { |cb|
157       if cb == "WITHOUT"
158         joins(:word).joins("LEFT JOIN `example_sentences` ON `words`.`id` = `↩
              example_sentences`.`word_id`").where(example_sentences: { id: nil ↩
              }).distinct
159       elsif cb == "WITH"
160         joins(:word).joins("INNER JOIN `example_sentences` ON `words`.`id` = `↩
              example_sentences`.`word_id`").distinct
161       end
162     }
163     scope :has_meaning, lambda { |cb|
164       if cb == "WITHOUT"
165         joins(:word).where(words: { meaning: [nil, ''] })
166       elsif cb == "WITH"
167         joins(:word).where.not(words: { meaning: [nil, ''] })
168       end
169     }
170     scope :is_foreign, lambda { |cb|
171       value = cb == "CHECKED" ? true : (cb == "UNCHECKED" ? false : nil)
172       joins(:word).where(foreign: value) unless cb == 'EMPTY'
173     }
174     scope :is_prototype, lambda { |cb|
175       value = cb == "CHECKED" ? true : (cb == "UNCHECKED" ? false : nil)
176       joins(:word).where(prototype: value) unless cb == 'EMPTY'
177     }
178     scope :is_compound, lambda { |cb|
179       value = cb == "CHECKED" ? true : (cb == "UNCHECKED" ? false : nil)
180       joins(:word).where(compound: value) unless cb == 'EMPTY'
181     }
182     scope :is_singularetantum, lambda { |cb|
183       value = cb == "CHECKED" ? true : (cb == "UNCHECKED" ? false : nil)
184       where(singularetantum: value) unless cb == 'EMPTY'
```

```
185     }
186     scope :is_pluraletantum, lambda { |cb|
187       value = cb == "CHECKED" ? true : (cb == "UNCHECKED" ? false : nil)
188       where(pluraletantum: value) unless cb == 'EMPTY'
189     }
190     scope :is_subjectless, lambda { |cb|
191       value = cb == "CHECKED" ? true : (cb == "UNCHECKED" ? false : nil)
192       where(subjectless: value) unless cb == 'EMPTY'
193     }
194     scope :is_perfect_haben, lambda { |cb|
195       value = cb == "CHECKED" ? true : (cb == "UNCHECKED" ? false : nil)
196       where(perfect_haben: value) unless cb == 'EMPTY'
197     }
198     scope :is_perfect_sein, lambda { |cb|
199       value = cb == "CHECKED" ? true : (cb == "UNCHECKED" ? false : nil)
200       where(perfect_sein: value) unless cb == 'EMPTY'
201     }
202     scope :is_irregular_declination, lambda { |cb|
203       value = cb == "CHECKED" ? true : (cb == "UNCHECKED" ? false : nil)
204       where(irregular_declination: value) unless cb == 'EMPTY'
205     }
206     scope :is_absolute, lambda { |cb|
207       value = cb == "CHECKED" ? true : (cb == "UNCHECKED" ? false : nil)
208       where(absolute: value) unless cb == 'EMPTY'
209     }
210     scope :is_irregular_comparison, lambda { |cb|
211       value = cb == "CHECKED" ? true : (cb == "UNCHECKED" ? false : nil)
212       where(irregular_comparison: value) unless cb == 'EMPTY'
213     }
214   end
215
216 end
```

Quellcode A.1 `lib/word_filter.rb`

Quellcode A.2 app/views/shared/_filter.html.haml

```
- filter = @filterrific.to_hash.select{|k, v| k.start_with?('search_', 'with_','is_', 'has_')}.reject{|k, v| k.start_with?('is_', 'has_') && 'EMPTY' == v}.map{|k, v| [k, map_relation(k, v)]}.reject{|e| e[1].nil?}
- unless filter.empty?
  .alert.alert-warning
    = succeed ':' do
      = t('filter.active_filter')
    - arr = []
    - for p in filter do
      - if p[0].start_with?('search_', 'with_')
        - arr << "#{t("filter.#{p[0].gsub /^(search_|with_)/, ''}")} (#{p[1].is_a?(Array) ? p[1][0] + ': ' + p[1].drop(1).join(", ") : p[1]})"
      - else
        - arr << "#{t("filter.#{p[0].gsub /^(is_|has_)/, ''}")} (#{t("filter.checkboxes.short.#{p[1]}")})"
    = arr.join ', '
    = link_to reset_filterrific_url, :class => 'alert-link ml-3' do
      %i.fa.fa-undo
      = t('filter.reset')
= form_for_filterrific @filterrific do |f|
  .row.align-items-end
    .col
      .row.col= t('filter.wordstarts')
      .row.col
        = f.text_field(:search_wordstarts, class: 'form-control filterrific-periodically-observed')
    .col
      .row.col= t('filter.wordcontains')
      .row.col
        = f.text_field(:search_wordcontains, class: 'form-control filterrific-periodically-observed')
    .col
      .row.col= t('filter.wordends')
      .row.col
        = f.text_field(:search_wordends, class: 'form-control filterrific-periodically-observed')
    .col
      .row.col= t('filter.wordconsists')
      .row.col
        = f.text_field(:search_wordconsists, class: 'form-control filterrific-periodically-observed')
    .col
```

```
35        .row
36          .col
37            -# FIXME: Hotfix to only show"advanced search options" on normal word↩
                types
38            - if ['Noun', 'Verb', 'Adjective'].include? @filterrific.model_class.↩
                to_s
39              %a.btn.btn-lg.btn-light{:href => '#collapseMoreFilter', 'data-↩
                  toggle' => 'collapse', :role => 'button', 'aria-expanded' => '↩
                  false', 'aria-controls' => 'collapseMoreFilter'}
40                %i.fa.fa-search-plus
41            = button_tag type: 'submit', class: 'btn btn-lg btn-primary', title: ↩
                t('filter.apply') do
42              %i.fa.fa-search
43    #collapseMoreFilter.collapse{:class => "#{'show' unless (filter.map{|k, v| k}↩
        - ['search_wordstarts', 'search_wordends', 'search_wordcontains', '↩
        search_wordconsists']).empty?}"}
44      .row.align-items-end
45        .col.col-12.col-md-6.col-lg-4
46          .row.col= t('filter.source')
47          .row.col
48            = f.select(:with_source,
49              Source.all.order(:name).map{|s|[s.name, s.id]},
50              { include_blank: true, selected: @filterrific.with_source },
51              { class: "form-control selectize-select"})
52        .col.col-12.col-md-6.col-lg-4
53          .row.col= t('filter.topic')
54          .row.col
55            = f.select(:with_topic,
56              Topic.all.order(:name).map{|s|[s.name, s.id]},
57              { include_blank: true, selected: @filterrific.with_topic },
58              { class: "form-control selectize-select"})
59        .col.col-12.col-md-6.col-lg-4
60          .row.col= t('filter.hierarchy')
61          .row.col
62            = f.select(:with_hierarchy,
63              Hierarchy.all.order(:name).map{|s|[s.name, s.id]},
64              { include_blank: true, selected: @filterrific.with_hierarchy },
65              { class: "form-control selectize-select"})
66      .row.align-items-end
67        .col.col-12.col-md-6.col-xl-4
68          .row.col= t('filter.phenomenons')
69          .row.col.input-group
70            = f.fields_for :with_phenomenons do |with_phenomenons|
```

```
71              .input-group-prepend.btn-group.btn-group-toggle{ 'data-toggle' => 'buttons' }
72                - (['AND', 'OR'].map{|e| [t("filter.and_or.short.#{e}"), e]}).each do |rb|
73                  - checked = @filterrific.with_phenomenons.try(:conjunction) == rb[1] || @filterrific.with_phenomenons.try(:conjunction).nil? && rb[1] == 'AND'
74                  = with_phenomenons.label :conjunction,
75                    { class: "btn btn-light#{checked ? ' active' : ''}" } do
76                    = with_phenomenons.radio_button(:conjunction,
77                      rb[1],
78                      checked: checked)
79                    = rb[0]
80              = with_phenomenons.select(:phenomenons,
81                Phenomenon.all.order(:name).map{|s|[s.name, s.id]},
82                { include_blank: true, selected: @filterrific.with_phenomenons.try(:phenomenons) },
83                { class: "form-control selectize-select", multiple: true})
84        .col.col-12.col-md-6.col-xl-4
85          .row.col= t('filter.strategies')
86          .row.col.input-group
87            = f.fields_for :with_strategies do |with_strategies|
88              .input-group-prepend.btn-group.btn-group-toggle{ 'data-toggle' => 'buttons' }
89                - (['AND', 'OR'].map{|e| [t("filter.and_or.short.#{e}"), e]}).each do |rb|
90                  - checked = @filterrific.with_strategies.try(:conjunction) == rb[1] || @filterrific.with_strategies.try(:conjunction).nil? && rb[1] == 'AND'
91                  = with_strategies.label :conjunction,
92                    { class: "btn btn-light#{checked ? ' active' : ''}" } do
93                    = with_strategies.radio_button(:conjunction,
94                      rb[1],
95                      checked: checked)
96                    = rb[0]
97              = with_strategies.select(:strategies,
98                Strategy.all.order(:name).map{|s|[s.name, s.id]},
99                { include_blank: true, selected: @filterrific.with_strategies.try(:strategies) },
100               { class: "form-control selectize-select", multiple: true})
101       .col.col-12.col-lg-6
102         .row.col= t('filter.keywords')
103         .row.col.input-group
```

```
104             = f.fields_for :with_keywords do |with_keywords|
105               .input-group-prepend.btn-group.btn-group-toggle{ 'data-toggle' => 'buttons' }
106                 - (['AND', 'OR'].map{|e| [t("filter.and_or.short.#{e}"), e]}).each do |rb|
107                   - checked = @filterrific.with_keywords.try(:conjunction) == rb[1] || @filterrific.with_keywords.try(:conjunction).nil? && rb[1] == 'AND'
108                   = with_keywords.label :conjunction,
109                     { class: "btn btn-light#{checked ? ' active' : ''}" } do
110                     = with_keywords.radio_button(:conjunction,
111                       rb[1],
112                       checked: checked)
113                     = rb[0]
114               = with_keywords.select(:keywords,
115                 [],
116                 { include_blank: true, selected: @filterrific.with_keywords.try(:keywords) },
117                 { class: "form-control selectize-select optgroup-select", multiple: true, 'data-lazy-load-url' => all_short_words_path,'data-selected'=> @filterrific.with_keywords.try(:keywords).to_json })
118       .row
119         .col-12.col-sm-6.col-md-4
120           .row.col= t('filter.consonant_vowel')
121           .row.col
122             = f.text_field(:search_consonant_vowel, class: 'form-control filterrific-periodically-observed', style: 'text-transform: uppercase;', placeholder: 'KVVK')
123         = filter_checkbox_field(f, @filterrific, :is_foreign, t('filter.foreign'))
124         = filter_checkbox_field(f, @filterrific, :is_prototype, t('filter.prototype'))
125         = filter_checkbox_field(f, @filterrific, :is_compound, t('filter.compound'))
126         = checkbox_field(f, :has_example_sentence, t('filter.example_sentence'), 'filter.checkboxes.short', ['WITH', 'WITHOUT', 'EMPTY'], @filterrific)
127         = checkbox_field(f, :has_meaning, t('filter.meaning'), 'filter.checkboxes.short', ['WITH', 'WITHOUT', 'EMPTY'], @filterrific)
128       .row
129         .col.col-12
130           %strong= t('filter.type_specific')
```

```
131         - [:singularetantum, :pluraletantum, :subjectless, :perfect_haben, :↩
                perfect_sein, :irregular_declination, :absolute, :↩
                irregular_comparison].each do |field|
132           = filter_checkbox_field(f, @filterrific, "is_#{field}".to_sym, t("↩
                  filter.#{field}")) if @filterrific.model_class.attribute_method? ↩
                  field
```

Quellcode A.2 `app/views/shared/_filter.html.haml`

Quellcode A.3 app/helpers/form_helper.rb

```
1  module FormHelper
2    def filter_checkbox_field(form, filterrific, name, label)
3      checkbox_field(form, name, label, 'filter.checkboxes.short', ['CHECKED', '↩
           UNCHECKED', 'NONE', 'EMPTY'], filterrific)
4    end
5    def inline_checkbox_field(form, name, t_keybase, t_keys, params)
6      capture_haml do
7        haml_tag(:div, class: 'btn-group btn-group-toggle', 'data-toggle' => '↩
             buttons') do
8          (t_keys.map{|e| [t("#{t_keybase}.#{e}").html_safe, e]}).each do |rb|
9            checked = (params.try(name) || params.try(:[], name)) == rb[1] || (↩
                 params.try(name) || params.try(:[], name)).blank? && rb[1] == '↩
                 EMPTY'
10           haml_concat(form.label(name,
11             { class: "btn btn-light#{checked ? ' active' : ''}" }) do
12             capture_haml do
13               haml_concat form.radio_button(name,
14                 rb[1],
15                 checked: checked)
16               haml_concat rb[0]
17             end.html_safe
18           end)
19         end
20       end
21     end
22   end
23   def checkbox_field(form, name, label, t_keybase, t_keys, params)
24     capture_haml do
25       haml_tag :div, class: 'col-12 col-sm-6 col-md-4 col-lg-3 col-xl-3' do
26         haml_tag :div, class: 'row col' do
27           haml_concat label
28         end
29         haml_tag :div, class: 'row col' do
30           haml_concat inline_checkbox_field(form, name, t_keybase, t_keys, ↩
                 params)
31         end
32       end
33     end
34   end
35   def input_number_field(form, name, id, label, text=nil, placeholder=nil, ↩
         explanation=nil)
36     label ||= ' '
```

```
37      capture_haml do
38        haml_tag :div, class: 'row form-group' do
39          haml_tag :div, class: 'col-sm-2 text-right-not-xs form-control-↩
                plaintext' do
40            if text.nil?
41                haml_concat form.label name, label.html_safe
42            else
43              haml_concat form.label name, label, class: 'sr-only'
44              haml_concat text.html_safe
45            end
46          end
47          haml_tag :div, class: 'col-sm-10' do
48            haml_concat form.number_field name, id: id, class: 'form-control', ↩
                  placeholder: placeholder
49          end
50          explain(explanation)
51        end
52      end
53    end
54
55    def input_text_field(form, name, id, label, text=nil, placeholder=nil, ↩
        explanation=nil)
56      label ||= ' '
57      capture_haml do
58        haml_tag :div, class: 'row form-group' do
59          haml_tag :div, class: 'col-sm-2 text-right-not-xs form-control-↩
                plaintext' do
60            if text.nil?
61                haml_concat form.label name, label.html_safe
62            else
63              haml_concat form.label name, label, class: 'sr-only'
64              haml_concat text.html_safe
65            end
66          end
67          haml_tag :div, class: 'col-sm-10' do
68            haml_concat form.text_field name, id: id, class: 'form-control', ↩
                  placeholder: placeholder
69          end
70          explain(explanation)
71        end
72      end
73    end
74
```

```
75    def input_pfix(form, name, word, disabled, label, text=nil)
76      if name == :prefix
77        atype = word.actable_type
78        elements = Prefix.all.select{|p| p.prefix_type == atype}.map{|p| [p.name,↩
              p.id]}.sort_by{|name, id| name}
79        selected = word.prefix_id
80      elsif name == :postfix
81        elements = Postfix.all.map{|p| [p.name, p.id]}.sort_by{|name, id| name}
82        selected = word.postfix_id
83      end
84      input_select(form, (name.to_s+'_id').to_sym, elements, label, text, nil, ↩
          selected, false, false, disabled)
85    end
86
87    def input_hierarchy(form, name, word, disabled, label, text=nil)
88        input_select(form, name, Hierarchy.order(:name).map{|h| [h.name, h.id]}, ↩
            label, text, nil, word.hierarchy_id, true, false, disabled)
89    end
90
91    def input_topics(form, name, word, disabled, label, text=nil)
92        input_select(form, name, Topic.order(:name).map{|t| [t.name, t.id]}, ↩
            label, text, nil, word.topics.map{|t| t.id}, true, true, disabled)
93    end
94
95    def input_select(form, name, elements, label, text=nil, explanation=nil, ↩
        selected={}, chosen=false, multiple=false, disabled=false, hsh = {})
96      capture_haml do
97        haml_tag :div, class: 'row form-group' do
98          haml_tag :div, class: 'col-sm-2 text-right-not-xs form-control-↩
              plaintext' do
99            if text.nil?
100             haml_concat form.label name, label.html_safe
101           else
102             haml_concat form.label name, label, class: 'sr-only'
103             haml_concat text.html_safe
104           end
105         end
106         haml_tag :div, class: 'col-sm-10', style: "#{'padding-top: 0.5rem;' if ↩
              chosen}" do
107           haml_concat form.select name,
108             elements,
109             { include_blank: true, selected: selected },
```

```
110             { class: "form-control#{' selectize-select' if chosen}", multiple: ↩
                    multiple, disabled: disabled }.merge(hsh)
111           end
112           explain(explanation, disabled)
113         end
114       end
115     end
116
117     def input_lazy_select(form, name, path, label, text=nil, explanation=nil, ↩
          selected={}, json_attribute='words', multiple=false, disabled=false, ↩
          optgroup = false, hsh = {})
118       capture_haml do
119         haml_tag :div, class: 'row form-group' do
120           haml_tag :div, class: 'col-sm-2 text-right-not-xs form-control-↩
                plaintext' do
121             if text.nil?
122               haml_concat form.label name, label.html_safe
123             else
124               haml_concat form.label name, label, class: 'sr-only'
125               haml_concat text.html_safe
126             end
127           end
128           haml_tag :div, class: 'col-sm-10' do
129             haml_concat form.select name,
130               [],
131               { include_blank: true, selected: selected },
132               { class: "form-control selectize-select#{' optgroup-select' if ↩
                    optgroup}", multiple: multiple, disabled: disabled , 'data-lazy↩
                    -load-url' => path, 'data-selected' => selected, 'data-lazy-↩
                    load-attribute' => json_attribute }.merge(hsh)
133           end
134           explain(explanation, disabled)
135         end
136       end
137     end
138
139     def input_bool(form, name, word, label, explanation=nil, disabled=false)
140       capture_haml do
141         haml_tag :div, class: 'row row-group' do
142           haml_tag :div, class: 'col-sm-2 text-right-not-xs form-control-↩
                plaintext' do
143             haml_concat form.label name, label.html_safe
144           end
```

```
145         haml_tag :div, class: 'col-sm-10' do
146           haml_tag :div, class: 'btn-group btn-group-toggle', 'data-toggle' => 'buttons' do
147             (['CHECKED', 'UNCHECKED', 'NONE'].map{|e| [t("filter.checkboxes.short.#{e}"), e]}).each_with_index do |rb, i|
148               checked = bool_to_int(word.public_send(name), [0,1,2]) == i
149               haml_concat(
150                 form.label(name,
151                            { class: "btn #{checked ? (rb[1] == 'NONE' ? 'btn-warning' : 'btn-light') + ' active' : 'btn-light'}" }) do
152                   haml_concat form.radio_button(name,
153                     [1, 0, 'NULL'][i],
154                     checked: checked)
155                   haml_concat rb[0]
156                 end
157               )
158             end
159           end
160           explain(explanation, disabled)
161         end
162       end
163     end
164   end
165
166   private
167   def explain(explanation, disabled=false)
168     unless explanation.nil? or disabled
169       haml_tag :small, class: 'form-text text-muted explanation' do
170         haml_concat explanation
171       end
172     end
173   end
174   def bool_to_int(bool, values=[1,0,2])
175     return values[2] if bool.nil?
176     return values[0] if bool
177     return values[1]
178   end
179 end
```

Quellcode A.3 app/helpers/form_helper.rb

Quellcode A.4 app/controller/nouns_controller.rb

```
1  class NounsController < ApplicationController
2    before_action :set_noun, only: [:show, :edit, :update, :destroy]
3    respond_to :html, :json
4    utf8_enforcer_workaround
5
6    # GET /nouns
7    # GET /nouns.json
8    def index
9      @filterrific = initialize_filterrific(
10       Noun,
11       params[:filterrific],
12     ) or return
13     @nouns = @filterrific.find.page(params[:page]).per(current_user.settings.↩
           page_size)
14   end
15
16   # GET /nouns/all.json
17   def index_all
18     respond_to :json
19     @filterrific = initialize_filterrific(
20       Noun,
21       params[:filterrific],
22       :persistence_id => false,
23     ) or return
24     @nouns = @filterrific.find
25     render :index
26   end
27   # GET /nouns/all/short.json
28   def index_all_short
29     respond_to :json
30     @filterrific = initialize_filterrific(
31       Noun,
32       params[:filterrific],
33       :persistence_id => false,
34     ) or return
35     @nouns = @filterrific.find
36     render :"index.short"
37   end
38
39   # POST /nouns/bulk_update
40   def bulk_update
41     if params[:word_ids].is_a?(Array) && params[:word_ids].length > 1
```

```
42        @nouns = Noun.find(params[:word_ids])
43        @nouns.each do |noun|
44          noun.update(bulk_params(noun))
45        end
46        redirect_to nouns_path, notice: t('notice.bulk_edit_saved', elements: ↩
              @nouns.size)
47      else
48        redirect_to nouns_path
49      end
50    end
51
52    # GET /nouns/1
53    # GET /nouns/1.json
54    def show
55    end
56
57    # GET /nouns/new
58    def new
59      @noun = Noun.new
60    end
61
62    # GET /nouns/1/edit
63    def edit
64    end
65
66    # POST /nouns
67    # POST /nouns.json
68    def create
69      @noun = Noun.new(noun_params)
70      points = count_changed(nil, noun_params)
71
72      respond_to do |format|
73        if @noun.save
74          format.html { redirect_created_with_points points, nouns_path, @noun }
75          format.json { render :show, status: :created, location: @noun }
76        else
77          format.html { render :new }
78          format.json { render json: @noun.errors, status: :unprocessable_entity ↩
                }
79        end
80      end
81    end
82
```

```
83    # PATCH/PUT /nouns/1
84    # PATCH/PUT /nouns/1.json
85    def update
86      points = count_changed(@noun, noun_params)
87
88      respond_to do |format|
89        if @noun.update(noun_params)
90          format.html { redirect_with_points points, @noun, nouns_path }
91          format.json { render :show, status: :ok, location: @noun }
92        else
93          format.html { render :edit }
94          format.json { render json: @noun.errors, status: :unprocessable_entity ↩
                }
95        end
96      end
97    end
98
99    # DELETE /nouns/1
100   # DELETE /nouns/1.json
101   def destroy
102     @noun.destroy
103     respond_to do |format|
104       format.html { redirect_destroy_with_points 0, @noun, nouns_url }
105       format.json { head :no_content }
106     end
107   end
108
109   private
110     # Use callbacks to share common setup or constraints between actions.
111     def set_noun
112       @noun = Noun.includes(:genus, :genus, :synonyms, :opposites, :keywords, :↩
              rimes, :topics).find(params[:id])
113     end
114
115     # Never trust parameters from the scary internet, only allow the white list↩
            through.
116     def noun_params
117       nil_params multval_params params.require(:noun).permit(:name, :syllables,↩
             :written_syllables, :prefix_id, :postfix_id, :meaning, :meaning_long↩
            , :plural, :genus_id, :genus_masculine_id, :genus_feminine_id, :↩
            genus_neuter_id, :hierarchy_id, :case_1_singular, :case_1_plural, :↩
            case_2_singular, :case_2_plural, :case_3_singular, :case_3_plural, :↩
            case_4_singular, :case_4_plural, :singularetantum, :pluraletantum, :↩
```

```
            prototype, :compound, :foreign, :keywords => [], :topics => [], :↩
            opposites => [], :synonyms => [], :rimes => [], :↩
            example_sentences_attributes => [:id, :sentence, :_destroy], :↩
            phenomenons => [], :strategies => [])
118     end
119
120     def bulk_params(noun)
121       return nil unless ['prefix_id', 'postfix_id', 'hierarchy_id', 'topics', '↩
              keywords', 'rimes', 'phenomenons', 'strategies'].include? params['↩
              bulk_type']
122       map = Hash.new
123       if ['topics', 'keywords', 'rimes', 'phenomenons', 'strategies'].include? ↩
              params['bulk_type']
124         map[params['bulk_type'].to_sym] = noun.try(params['bulk_type']).to_a
125         map[params['bulk_type'].to_sym] << params['bulk_value']
126       else
127         map[params['bulk_type'].to_sym] = params['bulk_value']
128       end
129       multval_params map
130     end
131 end
```

Quellcode A.4 app/controller/nouns_controller.rb

Quellcode A.5 app/views/nouns/_form.html.haml

```
- model = Noun
= form_with(model: noun, local: true) do |form|
  - if local_assigns[:edit]
    .float-right
      = form.submit t('helpers.titles.save', model: tm(model)), :class => 'btn btn-success'
  %h1= t("helpers.titles.#{local_assigns[:edit] ? (noun.new_record? ? 'new' : 'edit') : 'show'}", model: tm(model))
  - if noun.errors.any?
    #error_explanation
      %h2
        = pluralize(noun.errors.count, "error")
        = " prohibited this noun from being saved:"

      %ul
        - noun.errors.full_messages.each do |message|
          %li= message

  %fieldset{disabled: local_assigns[:edit].nil?}
    .row
      .col-md-6
        = input_text_field form, :name, :noun_name, ta(model, :name), nil, 'Elefant'
      .col-md-6
        = input_text_field form, :plural, :noun_plural, ta(model, :plural), nil, 'Elefanten'
    .row
      .col-md-6
        - text = capture do
          Manche Substantive kommen nur im Singular vor, Beispiele sind:
          %i Schnee, Vernunft, Gesundheit, Butter, Post.
          %a{:href=>'https://de.wikipedia.org/wiki/Singularetantum', :target=>'_blank'}
            %i.fa.fa-wikipedia-w
              Singularetantum
        = input_bool form, :singularetantum, noun, ta(model, :singularetantum), text, local_assigns[:edit].nil?
      .col-md-6
        - text = capture do
          Manche Substantive kommen nur im Plural vor, Beispiele sind:
          %i Ferien, Kosten, Leute.
```

```
36            %a{:href=>'https://de.wikipedia.org/wiki/Pluraletantum', :target=>'↩
                  _blank'}
37              %i.fa.fa-wikipedia-w
38                Pluraletantum
39          = input_bool form, :pluraletantum, noun, ta(model, :pluraletantum), ↩
                text, local_assigns[:edit].nil?
40      .row
41        .col-md-6
42          = input_text_field form, :syllables, :noun_syllables, ta(model, :↩
                syllables), nil, 'E-le-fant'
43        .col-md-6
44          = input_text_field form, :written_syllables, :noun_written_syllables, ↩
                ta(model, :written_syllables), nil, 'Ele|fant'
45      = render('shared/strategies_phenomenons', form: form, word: @noun, disabled↩
            : local_assigns[:edit].nil?)
46      .row
47        .col-md-6
48          = input_pfix form, :prefix, noun, local_assigns[:edit].nil?, 'Vorsilbe'
49        .col-md-6
50          = input_pfix form, :postfix, noun, local_assigns[:edit].nil?, 'Endung'
51
52      .row
53        .col-md-6
54          = input_bool form, :compound, noun, ta(model, :compound), nil, ↩
                local_assigns[:edit].nil?
55
56      .row
57        .col-md-6
58          = input_text_field form, :meaning, :noun_meaning, ta(model, :meaning), ↩
                nil, 'Kurzbedeutung (1-2 Wörter)', 'z.B. beim Wort "Bauer" könnte ↩
                eine Bedeutung "Schachfigur", eine andere "Landwirt" sein'
59        .col-md-6
60          = input_text_field form, :meaning_long, :noun_meaning_long, ta(model, :↩
                meaning_long), nil, 'längere Bedeutung'
61
62      .row
63        .col-md-6
64          = input_select form, :genus_id, Genus.all.map{|g| [g.name, g.id]}, ta(↩
                model, :genus), nil, "#{@noun.genus.nil? ? 'Ohne vorab ausgewähltes↩
                 Geschlecht können keine anderen Formen ausgewählt werden.':nil}", ↩
                @noun.genus_id, false, false, local_assigns[:edit].nil?
65        .col-md-6
66          = input_lazy_select form,
```

```
67            :genus_neuter_id,
68            all_short_nouns_path('filterrific[with_genus]' => 'NEUTER'), 'Form&↩
                  nbsp;(n)', nil, nil, @noun.genus_neuter_id, 'nouns', false, ↩
                  local_assigns[:edit].nil?
69        .row
70          .col-md-6
71            = input_lazy_select form,
72              :genus_masculine_id,
73              all_short_nouns_path('filterrific[with_genus]' => 'MASCULINUM'), '↩
                  Form (m)', nil, nil, @noun.genus_masculine_id, 'nouns', ↩
                  false, local_assigns[:edit].nil?
74          .col-md-6
75            = input_lazy_select form,
76              :genus_feminine_id,
77              all_short_nouns_path('filterrific[with_genus]' => 'FEMININUM'), 'Form↩
                   (f)', nil, nil, @noun.genus_feminine_id, 'nouns', false, ↩
                  local_assigns[:edit].nil?
78
79        .row
80          .col-md-6
81            = input_bool form, :prototype, noun, ta(model, :prototype), nil, ↩
                local_assigns[:edit].nil?
82          .col-md-6
83            = input_bool form, :foreign, noun, ta(model, :foreign), nil, ↩
                local_assigns[:edit].nil?
84        .row
85          .col-md-6
86            = input_hierarchy form, :hierarchy_id, @noun, local_assigns[:edit].nil↩
                ?, ta(model, :hierarchy)
87          .col-md-6
88            = input_topics form, :topics, @noun, local_assigns[:edit].nil?, ↩
                ta_pluralize(model, :theme)
89        .row
90          .col
91            %strong Fälle / Kasus
92        .row
93          .col
94            %strong 1. Fall / Nominativ
95            %i Wer oder was?
96        .row
97          .col-md-6
98            .row
99              %strong.col.text-center
```

```
100             Singular
101         .row
102           .col
103             =input_text_field form, :case_1_singular, :noun_case_1_singular, %w(der die das der/die der/das die/das der/die/das)[@noun.genus_id||7], nil, 'Elefant'
104       .col-md-6
105         .row
106           %strong.col.text-center
107             Plural
108         .row
109           .col
110             =input_text_field form, :case_1_plural, :noun_case_1_plural, 'die', nil, 'Elefanten'
111     .row
112       .col
113         %strong 2. Fall / Genitiv
114         %i Wessen?
115     .row
116       .col-md-6
117         .row
118           %strong.col.text-center
119             Singular
120         .row
121           .col
122             =input_text_field form, :case_2_singular, :noun_case_2_singular, %w(des der des des/der des der/des des/der)[@noun.genus_id||7], nil, 'Elefanten'
123       .col-md-6
124         .row
125           %strong.col.text-center
126             Plural
127         .row
128           .col
129             =input_text_field form, :case_2_plural, :noun_case_2_plural, 'der', nil, 'Elefanten'
130     .row
131       .col
132         %strong 3. Fall / Dativ
133         %i Wem?
134     .row
135       .col-md-6
136         .row
```

```
137             %strong.col.text-center
138               Singular
139           .row
140             .col
141               =input_text_field form, :case_3_singular, :noun_case_3_singular, %w↩
                      (dem der dem dem/der dem der/dem dem/der)[@noun.genus_id||7], ↩
                      nil, 'Elefanten'
142         .col-md-6
143           .row
144             %strong.col.text-center
145               Plural
146           .row
147             .col
148               =input_text_field form, :case_3_plural, :noun_case_3_plural, 'den',↩
                       nil, 'Elefanten'
149       .row
150         .col
151           %strong 4. Fall / Akkusativ
152           %i Wen oder was?
153       .row
154         .col-md-6
155           .row
156             %strong.col.text-center
157               Singular
158           .row
159             .col
160               =input_text_field form, :case_4_singular, :noun_case_4_singular, %w↩
                      (den die das den/die den/das die/das den/die/das)[@noun.↩
                      genus_id||7], nil, 'Elefanten'
161         .col-md-6
162           .row
163             %strong.col.text-center
164               Plural
165           .row
166             .col
167               =input_text_field form, :case_4_plural, :noun_case_4_plural, 'die',↩
                       nil, 'Elefanten'
168
169       = render('shared/example_sentences', form: form, disabled: local_assigns[:↩
            edit].nil?)
170
171       = render('shared/relations', form: form, word: @noun, disabled: ↩
            local_assigns[:edit].nil?, words_list: Word.includes(:actable))
```

```
172
173     - if local_assigns[:edit]
174       .row.form-group
175         .col
176           = form.submit t('helpers.titles.save', model: tm(model)), :class => '↩
                btn btn-success'
```

Quellcode A.5 `app/views/nouns/_form.html.haml`

Quellcode A.6 app/assets/javascripts/initializers.coffee

```
$ ->
  # enable chosen js
  $('.chosen-select').chosen
    allow_single_deselect: true
    no_results_text: t.chosen.no_results_text
    width: '100%'
    placeholder_text_multiple: t.chosen.placeholder_text_multiple
    placeholder_text_single: t.chosen.placeholder_text_single
  # enable selectize
  $('select:not([data-lazy-load-url]).selectize-select').selectize
      plugins: ['remove_button']
      selectOnTab: true
      valueField: 'id'
      labelField: 'name'
      searchField: 'name'

  $('select[data-lazy-load-url].selectize-select:not(.optgroup-select)').each (↩
      index, element) ->
    $(element).selectize
      plugins: ['remove_button']
      preload: true
      maxOptions: 1000
      selectOnTab: true
      valueField: 'id'
      labelField: 'name'
      searchField: 'name'
      load: (query, callback) ->
        if (query.length)
          return callback()
        $.ajax
          url: $(element).data('lazyLoadUrl')
          type: 'GET'
          cache: false
          error: () ->
            callback()
          success: (res) ->
            callback(res[$(element).data('lazyLoadAttribute')])
            if $(element).data('selected')
              for sel in $(element).data('selected')
                $(element)[0].selectize.addItem(sel)

```

```
41    $('select[data-lazy-load-url].selectize-select.optgroup-select').each (index,↩
          element) ->
42      $(element).selectize
43        plugins: ['remove_button', 'optgroup_columns']
44        preload: true
45        maxOptions: 1000
46        selectOnTab: true
47        valueField: 'id'
48        labelField: 'name'
49        searchField: 'name'
50        optgroupField: 'actable_type'
51        optgroupLabelField: 'name'
52        optgroupValueField: 'type'
53        optgroups: [
54          {type: 'Noun', name: t.selectize.nouns}
55          {type: 'Verb', name: t.selectize.verbs}
56          {type: 'Adjective', name: t.selectize.adjectives}
57        ]
58        optgroupOrder: ['Noun', 'Verb', 'Adjective']
59        lockOptgroupOrder: true
60        onItemAdd: (value, $item) ->
61          if $(element).hasClass('search-words')
62            window.location.href = '/words/' + value + '/edit'
63        load: (query, callback) ->
64          if (query.length)
65            return callback()
66          $.ajax
67            url: $(element).data('lazyLoadUrl')
68            type: 'GET'
69            error: () ->
70              callback()
71            success: (res) ->
72              callback(res.words)
73              if $(element).data('selected')
74                for sel in $(element).data('selected')
75                  $(element)[0].selectize.addItem(sel)
76
77    # enable bootstrap-popover
78    $('[data-toggle="popover"]').popover();
79    # enable bulk-edit
80    $("select[data-drop=dropup]").next().find(".chosen-drop").addClass("dropup");
81    $("select[data-drop=dropup]").next().find(".chosen-single").addClass("dropup"↩
          );
```

```
82    $('.bulk_edit_all').click ->
83      check = this.checked
84      $('.bulk_edit').prop('checked', check)
85      bulk()
86    bulk = ->
87      check_count = $('.bulk_edit:checked').length
88      if check_count > 1
89        $(".bulk-edit-form > *").show()
90      else
91        $(".bulk-edit-form > *").hide()
92    $('.bulk_edit').click -> bulk()
93    bulk()
94    $('.bulk-edit-form').submit ->
95      $('.bulk-edit-form input:checked').remove()
96      selected_items = $('.bulk_edit:checked').clone()
97      $('.bulk-edit-form').append(selected_items)
98      return true
99    bulk_confirm = ->
100     count = $('.bulk_edit:checked').length
101     type = $('select[name=bulk_type]').children("option:selected").text()
102     value = $('select[name=bulk_value]').children("option:selected").text()
103     verb = 'gesetzt'
104     if ['topics', 'keywords', 'rimes', 'phenomenons', 'strategies'].includes(↩
          type)
105       verb = 'hinzugefügt'
106     str = 'Sicher, dass für ' + count + ' Wörter "' +     value + '" als ' + ↩
          type + ' ' + verb + ' wird?'
107     console.log str
108     $('.bulk-edit-form button[type=submit]').data('confirm', str)
109   $('select[name=bulk_type]').change -> bulk_confirm()
110   $('select[name=bulk_value]').change -> bulk_confirm()
111   $('select[data-dynamic-selectable-url][data-dynamic-selectable-target]').↩
        dynamicSelectable()
112   $('select[data-dynamic-data-selectable-target]').dynamicDataSelectable()
113   #$('select[data-lazy-load-url]').lazyLoadSelect()
```

Quellcode A.6 app/assets/javascripts/initializers.coffee

Quellcode A.7 app/models/source.rb

```
class Source < ApplicationRecord
    validates :name, :presence => true
    has_and_belongs_to_many :words

    require 'csv'
    require 'charlock_holmes'
    def self.import(file, source)
        return nil if file.nil? or source.nil?
        count = 0
        updated = 0
        comment = file.original_filename.gsub(/^.+?_|\.csv/, '') + '_' + Date.current.strftime('%Y-%m-%d')
        data = []
        contents = File.read(file.path)
        detection = CharlockHolmes::EncodingDetector.detect(contents)
        CSV.foreach(file.path, headers: true, col_sep: ',', encoding: detection[:encoding]) do |row|
            return nil if word_type(row[0]).nil? or (row[1].nil? or row[1].empty?)
            hash = Hash.new
            hash[:type] = word_type row[0]
            hash[:data] = Hash.new
            hash[:data][:name] = row[1].strip
            hash[:data][:meaning] = row[2].strip unless row[2].nil?
            hash[:data][:genus_id] = genus row[4] if hash[:type] == :noun
            last = data.last
            if !last.nil? and last[:data][:name] == hash[:data][:name] and last[:data][:meaning] == hash[:data][:meaning] and last[:type] == hash[:type]
                last[:topics].concat (row[3]||'').split(';')
            else
                hash[:topics] = (row[3]||'').split(';')
                data << hash
            end
        end
        data.each do |w|
            if w[:data][:meaning].blank?
                words = w[:type].to_s.classify.constantize.where(name: w[:data][:name])
            else
                words = [w[:type].to_s.classify.constantize.find_by(name: w[:data][:name], meaning: w[:data][:meaning])]
```

```
36              end
37              words = [nil] if words.count == 0
38              words.each do |word|
39                  if word.nil?
40                      word = w[:type].to_s.classify.constantize.create! w[:data]
41                      count += 1
42                  else
43                      if w[:type] == :noun
44                          word.genus_id ||= w[:data][:genus_id]
45                      end
46                      updated += 1
47                  end
48
49                  w[:topics].uniq.each do |t|
50                      begin
51                          word.topics << Topic.find_or_create_by(name: t.strip)
52                      rescue ActiveRecord::RecordNotUnique
53                          logger.debug "Duplicate topic '#{t.strip}' for #{w[:↩
                                type]} '#{word.name}'"
54                      end
55                  end
56                  begin
57                      word.sources << source
58                  rescue ActiveRecord::RecordNotUnique
59                      updated -= 1
60                      logger.debug "Duplicate source '#{source.name}' for #{w[:↩
                            type]} '#{word.name}'"
61                  end
62              end
63          end
64          { new: count, updated: updated }
65      end
66
67      private
68          def self.word_type(type)
69              case type.strip.downcase
70              when 'adjektiv', 'adjektive'
71                  return :adjective
72              when 'substantiv', 'substantive', 'nomen'
73                  return :noun
74              when 'verb', 'verben'
75                  return :verb
76              else
```

```
77                  return nil
78              end
79          end
80          def self.genus(genus)
81              if genus.nil?
82                  return nil
83              end
84              case genus.strip.downcase
85              when 'maskulinum'
86                  return 0
87              when 'femininum'
88                  return 1
89              when 'neutrum'
90                  return 2
91              else
92                  return nil
93              end
94          end
95  end
```

Quellcode A.7 `app/models/source.rb`

Quellcode A.8 app/controller/sources_controller.rb

```
1  class SourcesController < ApplicationController
2    before_action do
3        redirect_to root_path unless current_user.is_admin?
4    end
5    before_action :set_source, only: [:show, :edit, :update, :destroy]
6
7    # GET /sources
8    # GET /sources.json
9    def index
10     @sources = Source.all
11   end
12
13   # GET /sources/1
14   # GET /sources/1.json
15   def show
16   end
17
18   # GET /sources/new
19   def new
20     @source = Source.new
21   end
22
23   # GET /sources/1/edit
24   def edit
25   end
26
27   # POST /sources
28   # POST /sources.json
29   def create
30     if params[:source][:file].nil?
31       redirect_to new_source_path, alert: 'Es wurde keine Quelldatei ausgewählt↩
             .'
32       return
33     end
34     file = params[:source].delete :file
35     @source = Source.new(source_params)
36
37     if @source.save
38       count = Source.import(file, @source)
39       if count.nil?
40           @source.delete
```

```
41            redirect_to new_source_path, alert: "Keine Wörter importiert, Fehler ↩
                  in der Quelldatei \"#{file.original_filename}\"."
42          else
43            redirect_to sources_path, notice: "#{count[:new]} neue Wörter ↩
                  importiert und #{count[:updated]} vorhandene Wörter verknüpft aus↩
                   der Datei \"#{file.original_filename}\"!"
44          end
45        else
46          render :new
47        end
48      end
49
50      # PATCH/PUT /sources/1
51      # PATCH/PUT /sources/1.json
52      def update
53        respond_to do |format|
54          if @source.update(source_params)
55            format.html { redirect_to @source, notice: 'Source was successfully ↩
                  updated.' }
56            format.json { render :show, status: :ok, location: @source }
57          else
58            format.html { render :edit }
59            format.json { render json: @source.errors, status: :↩
                  unprocessable_entity }
60          end
61        end
62      end
63
64      # DELETE /sources/1
65      # DELETE /sources/1.json
66      def destroy
67        @source.words.each do |w|
68            if w.sources.count <= 1
69                w.destroy
70            end
71        end
72        @source.destroy
73        respond_to do |format|
74          format.html { redirect_to sources_url, notice: 'Source was successfully ↩
                destroyed.' }
75          format.json { head :no_content }
76        end
77      end
```

```
78
79    private
80      # Use callbacks to share common setup or constraints between actions.
81      def set_source
82        @source = Source.find(params[:id])
83      end
84
85      # Never trust parameters from the scary internet, only allow the white list↩
            through.
86      def source_params
87        params.require(:source).permit(:name, :author, :licence, :source_url, :↩
              comment, :file)
88      end
89  end
```

Quellcode A.8 app/controller/sources_controller.rb

Quellcode A.9 app/models/user.rb

```
1  class User < ApplicationRecord
2      acts_as_token_authenticatable
3
4      has_many :team_members
5      has_many :teams, :through => :team_members
6
7      include RailsSettings::Extend
8      def page_size
9          settings.page_size
10     end
11     def page_size=(ps)
12         settings.page_size = ps
13     end
14
15     # Include default devise modules. Others available are:
16     # :confirmable, :lockable, :timeoutable and :omniauthable
17     devise :invitable, :database_authenticatable,
18            # :registerable,
19            # :recoverable,
20            # :rememberable,
21            # :trackable,
22            # :validatable,
23            # :confirmable,
24            :omniauthable, omniauth_providers: [:shibboleth]
25
26     def self.from_omniauth(auth)
27       user = find_by_provider_and_uid(auth.provider, auth.uid)
28       if user
29         user.given_name = auth.info.given_name if user.given_name.blank?
30         user.family_name = auth.info.family_name if user.family_name.blank?
31         user.email = auth.info.email if user.email.blank?
32         user.save
33       end
34       user
35     end
36
37     def is_admin?
38         self.admin
39     end
40     def can_write?
41       self.write || self.admin
42     end
```

```
43
44      include Gravtastic
45      gravtastic :secure => true, :size => 30, :default => 'retro'
46
47      require 'csv'
48      def self.import(file, write=false, comment=nil)
49          count = 0
50          updated = 0
51          if comment.nil? || comment.strip.empty?
52              comment = file.original_filename.gsub(/^.+?_|\.csv/, '') + '_' + ↩
                    Date.current.strftime('%Y-%m-%d')
53          end
54          comment.strip!
55          CSV.foreach(file.path, headers: true, col_sep: ';', encoding: 'UTF-8') ↩
                do |row|
56              if User.exists?(uid: row[5])
57                  u = User.find_by(uid: row[5])
58                  unless u.is_admin?
59                      u.comment = comment
60                      u.write = write
61                      u.save
62                      updated = updated + 1
63                  end
64                  next
65              end
66
67              hash = Hash.new
68              hash[:uid] = row[5]
69              hash[:email] = "#{row[5]}@uni-bremen.de"
70              hash[:given_name] = row[2]
71              hash[:family_name] = row[3]
72              hash[:provider] = 'shibboleth'
73              hash[:points] = 0
74              hash[:write] = write
75              hash[:comment] = comment
76              User.create! hash
77              count = count + 1
78          end
79          { new: count, updated: updated }
80      end
81  end
```

Quellcode A.9 app/models/user.rb

Quellcode A.10 app/controller/words_controller.rb

```
1  class WordsController < ApplicationController
2    utf8_enforcer_workaround
3
4    # GET /words/all/short.json
5    def index_all_short
6      respond_to :json
7      @words = Word.all.order("LOWER(`name`) ASC")
8      render :"index.short"
9    end
10
11   # GET /words/edit
12   def edit_redirect
13     w = Word.find(params[:id])
14     case w.actable_type
15     when 'Noun'
16       redirect_to edit_noun_path(w.actable.id)
17     when 'Adjective'
18       redirect_to edit_adjective_path(w.actable.id)
19     when 'Verb'
20       redirect_to edit_verb_path(w.actable.id)
21     else
22       redirect_to root_path
23     end
24   end
25 end
```

Quellcode A.10 app/controller/words_controller.rb

A.9 Sonstige Materialien

A.9.1 Datenbankstruktur, Initialversion

Inhalt	Name		Typ	Beispiel	
ID	Word_ID		Longint		
Wort	Word		Text		
Wortarten (STTS?/Montessori)	Wordtype_ID_f		Longint	ADV = Adverb	Ermöglicht Wortartensymbole u.ä. http://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/TagSets/stts-table.html http://lernmatrix.com/Downloads/Wortartensymbole.jpg
Hierarchische Struktur	Hierarchy_ID_f		Longint		wordnet/imagenet
Stichwörter		Mehrere			
Wortverwandtschaften		Mehrere			Zug,ziehen, Umzug
Wortfelder					Zug, ICE, Eisenbahn, Schranke, fahren
Artikel bestimmt					der die das
Artikel unbestimmt					ein, eine
Geschlecht	Gender_ID_f		Int		Ermöglicht Farbkodierung
Weibliche Form		Mehrere		die Pilotin	

Neutrale Form					
Männliche Form				der Pilot	
Vorsilben	Prefix_ID_f		Longint		
Endungen	Suffix_ID_f				
Silben					
Zahl der Silben			Int		
Anlaute	Anlaut_ID_f				
Laute					
Beispielsätze	SampleSentence		Text		
Wortschwierigkeit (vgl. Kieler Lese- und Schreibaufbau)		Berechnet			
Leichte Sprache Ranking					
Quizfragen		Mehrere			
nach Rechtschreibproblemen	Problem_ID	Mehrere			
Lupen auf Schwierigkeiten		Mehrere			

benötigte Strategien		Mehrere			
Ausnahmen		Mehrere			
Alternativen		Mehrere			
Gegenteile		Mehrere			
Reimwörter		Mehrere			
Braille	Braille				
Link zum Gebärdenlexikon		Mehrere			
falsche Schreibweisen		Mehrere			
Wortstamm	Wortstamm_ID_f				
Verben					
Zeitformen					
Beugung					
Benötigte Präpositionen				sich verlassen auf verlassen	
Erforderliche Fälle				Das Geschenk kaufen	

				Dem Menschen helfen	
Substantive					
Fälle					
Singular/Plural					
Geschlecht	Geschlecht_ID_f				
Farbkodierung					
Adjektive					
Steigerungen		Mehrere			
Funktionswort		Ja/nein		Artikel, Hilfsverben, Konjunktionen, Präpositionen, und Pronomen	
Audiodateien		Mehrere			
Sprache		Mehrere			
Bild		Mehrere			

Fehlermeldung				Rückmeldemöglichkeit für Fehler in der Datenbank
Status			Ungeprüft/geprüft/Fehlermeldung	Eingabestatus

A.10 Digitaler Anhang – CD

Als weiteren Anhang dieser Bachelorarbeit habe ich eine CD beigefügt. Die CD enthällt eine digitale Fassung der Bachelorarbeit und der darin verwendeten Grafiken, sowie den zum Abgabezeitpunkt aktuellen Stand des implementierten Systems.