Maximilian Schafzahl

BSc

NLP Webservice zur Textkorrektur integriert in eine moderne, serviceorientierte Webanwendung

oekorr

Masterarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium Softwareentwicklung-Wirtschaft

Technische Universität Graz

Fakultät für Informatik und Biomedizinische Technik

Institute for Interactive Systems and Data Science

Institutsvorstand: Univ.-Prof. Dr. Stefanie Lindstaedt

Betreuer: Assoc.Prof. Dipl.-Ing. Dr.techn. Denis Helic

Graz, Juli 2019

Danksagung

Ich danke meiner Familie für die großartige Unterstützung.

Allen voran danke ich meinen Eltern Maria und Wolfgang für die Vermittlung wichtiger Werte und die Weichenstellung in frühen Jahren, um an meiner Ausbildung zu arbeiten und mir damit ein selbstbestimmtes Leben zu ermöglichen.

Ich danke meinen Geschwistern Lena, Paul und Nikolaus, die mich laufend unterstützen sowie engen Freunden für motivierende Gespräche.

Für die Ermöglichung und Betreuung dieser Arbeit bedanke ich mich bei Prof. Denis Helic und Prof. Rudolf Muhr.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Graz, am …….

(Unterschrift)

Abstract

Goal of this thesis is it to build a prototype, which allows a software application to convert texts written in German language commonly used in Germany into German language texts commonly used in Austria. Due to the vast amount of texts, journalists are not able to correct every text they reprint from German media to Austrian media. Austrian readers are not used to German vocabulary and cannot comprehend it entirely. There is also a chance that Austrian language gets changed and vocabulary is lost as result of the influence of these texts.

In the first chapter linguistic technology is introduced. Texts need to be tokenized and part-of-speech tagged. Afterwards they are lemmatized to do dictionary lookups.

In the second chapter, software engineering is in focus to build a modern web API, which needs to be designed and secured properly. A Single Page Application is developed to please users with a functional graphical user interface.

The resulted prototype is useful and additional vocabulary can be added to improve the quality of conversions.

Kurzfassung

Mit dieser Arbeit soll ein Prototyp zur Textkorrektur deutschländisch Deutscher Texte in österreichisch Deutscher Texte entwickelt werden. Journalisten und Redakteure sind durch die große Menge an Texten nicht in der Lage, alle Artikel, die von Bundesdeutschen Medien übernommen werden manuell an den in Österreich gebräuchlichen Wortschatz anzupassen. Dadurch kann sich der Leser schlechter mit den Texten identifizieren und es geht der österreichische Wortschatz verloren. Automatisierte Unterstützung würde dies erleichtern.

In den ersten Kapiteln werden linguistische Technologien eingeführt. Vor allem die flache Satzverarbeitung findet hier Verwendung. Das Auftrennen der Texte, Part-of-Speech Tagging zum Bestimmen der Wortarten und zurückführen zu den Stammformen ist eine Herausforderung. Danach werden Korrekturtechnologien besprochen.

Im zweiten Teil der Arbeit liegt der Fokus auf der Softwareentwicklung eines modernen Webservices als Web API. Dazu werden gute Designs und Sicherheitsmechanismen besprochen. Der Einsatz einer Single Page Application, welche das Webservice nutzt, stellt dem Benutzer schlussendlich eine funktionale moderne grafische Oberfläche bereit.

Der entwickelte Prototyp kann benutzt und selbstständig durch weitere Tautonismen erweitert werden, um die Korrekturqualität zu verbessern.

Inhalt

Glossar

deutschländisch	binnendeutsch, wird besonders in der österreichischen und schweizerischen Sprachwissenschaft gebraucht, zur Differenzierungen des Deutschen Sprachgebrauchs
Teutonismus	ein sprachlicher Ausdruck, der typisch/charakteristisch für die Varietäten des deutschen Deutsch ist
Antezedens	Ursache, Grund
Lexikografie	Erstellung von Wörterbüchern oder lexikalischen Datenbanken
Lemma	bezeichnet in Linguistik und Lexikografie die Grundform eines Wortes
Austriazismen	spezielle in Österreich gebräuchliche Wörter
Österreichisches Deutsch	Absichtlich mit großem Anfangsbuchstaben da als Eigenwort verwendet

1 Einleitung

1.1 Motivation

Die Deutsche Sprache gehört zu den plurizentrischen Sprachen und wird in Österreich primär in der Varietät des Österreichischen Deutsch verwendet. Zu dieser Varietät entstand 1951 das „Österreichische Wörterbuch“ und es wurde damit eine Sammlung der in Österreich verwendeten Varianten geschaffen. Fortan wurde dieses in Österreich als amtlich gültiges Regelwerk anerkannt und nahm Einzug als Verwaltungs- und Unterrichtssprache. Dieses wird in Österreich als einheitlich gültige Rechtschreibung üblicherweise verwendet. Das Österreichische Deutsch hat somit nicht nur einen Sonderwortschatz, sondern wird auch von eigenen Stilen und Grammatiken geprägt und zeigt charakteristische Eigenheiten in der gesprochenen sowie auch geschriebenen Sprache. (1)

Im Alltag werden Austriazismen aber oft als Dialektform oder Umgangssprache angesehen und deshalb als falsch oder fehlerhaft gewertet. Untersuchungen zeigen, dass die Einstellung zum Österreichischen Deutsch von Vorbehalten geprägt ist und es in seiner Gleichwertigkeit dem deutschländischen Deutsch untergeordnet wird.

Diese Abwertung dürfte durch ein fehlendes Bewusstsein der kulturellen Identität des Österreichischen Deutsch begründet sein sowie die Unsicherheiten zur Sprache in Österreich auf Uninformiertheit beruhen.

In einer Untersuchung des Vorkommens von bundesdeutschem Wortgut in der österreichischen Pressesprache wurde festgestellt, dass oftmals Teutonismen die österreichischen Varianten ersetzen. Meist werden deutschländisch-deutsche Wörter nicht nur eingesetzt um Bezeichnungslücken zu füllen, sondern ein österreichisches Wort damit ersetzt. Dabei werden sogar Worte aus der bundesdeutschen Umgangssprache verwendet und diese als standardnäher, der Pressesprache angepasster empfunden als eigene standardsprachliche oder umgangssprachliche Wörter. Manchmal liegt es auch daran, dass Texte von ausländischen Redaktionen weiterverarbeitet werden und es den Redaktionen an Zeit fehlt, um die Texte an österreichische Standardsprache anzupassen.

1.2 Aufgabenstellung

Da es einige Sammlungen mit Tautonismen gibt, wird im Rahmen dieser Masterarbeit ein Prototyp zur automatisierten Auffindung und Textübersetzung von Bundesdeutschen in Österreichisch Deutsche Texte entwickelt. Das technische Hauptaugenmerk liegt dabei auf der Entwicklung eines modernen Webservices, welches die Verarbeitung und Übersetzung von Texten übernimmt.

Gemeinsam mit Prof. Rudolf Muhr der Forschungsstelle Österreichisches Deutsch der Karl-Franzens-Universität Graz wird ein Korrektursystem – *oekorr* – entworfen, um damit das Auffinden von Tautonismen zu vereinfachen.

Im ersten Schritt wird in einer Literaturrecherche nach verwendbaren Tools und Bibliotheken gesucht, die diese Aufgabe unterstützen können. Nach einem generellen Einblick in die Computerlinguistik und deren formaler Grundlagen ist die flache Satzverarbeitung für diese Arbeit relevant. Um Texte korrigieren zu können, müssen diese vorab analysiert werden. Nach einer Kategorisierung der einzelnen Worte müssen dafür Korrekturvorschläge gemacht werden.

Die Computerlinguistik umfasst ein sehr breites Spektrum an Forschungs- und Anwendungsgebieten und es müssen aus einem großen Angebot an Forschungsergebnissen Bibliotheken ausgewählt und evaluiert werden. Da es vor allem Tools zur Verarbeitung von Englischer Sprache gibt und in dieser Arbeit aber ausschließlich Deutsche Texte verarbeitet werden, müssen auch experimentelle Anwendungen in Betracht gezogen werden.

Nach der erfolgreichen Sprachverarbeitung werden diese Techniken in eine Webanwendung bzw. in ein derer zugrundeliegendes Webservice implementiert. Diese API soll einerseits von der Webanwendung benutzt werden aber auch als externe Schnittstelle angeboten werden können. Die Nutzungsmöglichkeiten dieser Schnittstelle werden beschrieben.

1.3 Gliederung

Im ersten Teil dieser Arbeit werden die technischen Hintergründe beschrieben beginnend mit der Sprachverarbeitung. Hierbei liegt der Fokus am Finden von Tokens, deren Klassifizierung und Finden der jeweiligen Lemmata. Danach wird die Funktionsweise von RESTful Webservices beschrieben und AngularJS als Frontendframework eingeführt.

Im zweiten Teil wird auf die Umsetzung der Softwarelösung eingegangen und deren Funktionsweise nachvollziehbar erklärt.

2 Technischer Hintergrund

2.1 Computerlinguistik und Sprachverarbeitung

Der Anwendungs- und Wissenschaftsbereich der Computerlinguistik ist der Informatik und Linguistik angesiedelt und überschneidet sich. Sie beschäftigt sich mit der maschinellen Verarbeitung von natürlicher Sprache. Seit der 50er Jahre des letzten Jahrhunderts wurden mit Wissen aus Informatik und Linguistik neue eigenständige Methoden für die Verarbeitung gesprochener und geschriebener Sprache entwickelt.

In unserer Informationsgesellschaft wächst der Einfluss von durch Wissen der Computerlinguistik entwickelter Software laufend und diese Programme sind aus dem Alltag nicht mehr wegzudenken. Bei täglichen, scheinbar simplen Websuchen benutzt man Sprach- bzw. Textverarbeitungstechniken. Das können morphologische Prozesse zur Rechtschreibkorrektur und grammatische Analysen sein, aber auch statistische Informationen zur Häufigkeitsanalyse und Lexikographie zur Mitberücksichtigung von Synonymen. Diese Prozesse beeinflussen das Suchergebnis und erhöhen dessen Treffsicherheit und Relevanz enorm.

Selbst wenn man kaum Computer benutzt kommt man mit Computerlinguistik in Kontakt. Sei es beim Lesen einer Bedienungsanleitung, die halbautomatisiert übersetzt wurde, bei der Sprachnavigation durch eine Telefonhotline oder der Verwendung von Chatbots. Dies zeigt sehr deutlich, welchen gesellschaftlichen Einfluss und Wichtigkeit Sprachverarbeitung hat und weshalb weitere Forschung und Entwicklung sowie die Erstellung weiterer Anwendungen nützlich ist. Es erleichtert den Umgang mit Maschinen und Computern, hilft Sprachbarrieren zu überwinden und erschließt neue Informationsquellen.

(2)

2.1.1 Die Wissenschaft

Die Computerlinguistik beschäftigt sich mit der Verarbeitung natürlicher Sprache sowohl in geschriebener als auch gesprochener Form. Klar abzugrenzen ist natürliche Sprache zu Plansprachen und formalen Sprachen.

Natürliche Sprache:

In der Sprachwissenschaft bezeichnet man als Natürliche Sprache eine von Menschen gesprochene (bzw. gebärdete Sprache), die historisch (natürlich) entstanden ist. Sie wird primär zur mündlichen und schriftlichen Kommunikation verwendet und weist strukturelle und lexikalische Unschärfe und Uneindeutigkeit auf.

Formale Sprache:

Im Unterschied zu Natürlicher Sprache ist Formale Sprache eine abstrakte Sprache, bei der nicht die natürliche, menschliche Kommunikation im Vordergrund steht, sondern die mathematische Verwendung. Eine formale Sprache besteht aus Zeichenketten (Strings) und Symbolen. Diese eignen sich zur (mathematisch) präzisen Beschreibung von Zeichenketten. Damit werden Formate und Programmiersprachen beschrieben.

Nach Jan Amtrup unterscheidet die Computerlinguistik vier Auffassungen:

- Computerlinguistik als Teildisziplin der Linguistik, die sich mit berechnungsrelevanten Aspekten von Sprache und Sprachverarbeitung beschäftigt, ohne diese unmittelbar in Software zu realisieren. (z.B. Grammatikformalismen entwickeln)
- Computerlinguistik als Disziplin für die Entwicklung von Programmen und die Linguistische Datenverarbeitungen (Sprachkorpora). Diese ist informatikgetrieben.
- Computerlinguistik als maschinelle Verarbeitung natürlicher Sprache und die Untersuchung natürlichsprachlicher Phänomene, welche eng mit Künstlicher Intelligenz und Kognitionswissenschaften zusammenhängt.
- Computerlinguistik als praxisorientierte Entwicklung von Sprachsoftware.

Diese vier Bereiche sind nicht klar voneinander getrennt, sondern greifen ineinander. Es beginnt mit der Entwicklung von Methoden um die natürliche Sprache Operationalisieren zu können. Daraus werden dann umfassende Sprachkorpora aufgebaut, welche dann später für die Entwicklung von Anwendungen verwendet werden können.

In weiterer Folge beschäftigen wir uns hier primär mit der letztgenannten Auffassung zur Entwicklung einer Sprachsoftware.

Von der Wissenschaft der Linguistik übernimmt die Computerlinguistik grundlegende Termini und Differenzierungen. Die Strukturierung der Computerlinguistik bedient sich der Linguistik und verwendet folgende Teilgebiete:

Phonetik und Phonologie:

In der Computerlinguistik zur Erkennung und Produktion gesprochener Sprache. Dabei werden artikulatorische Merkmale und Lautstruktur natürlicher Sprache untersucht.

Dieb	Diebe
/Diep/	/Diebe/

Tabelle 2-1 Beispiel Auslautverhärtung (2)

Morphologie:

Theorie zur Bildung und Struktur von Wörtern. Hier werden lexikalische Wurzeln der Wörter und Prozesse, die für die Erscheinungsform, Verwendung und Bedeutung eines Wortes verantwortlich sind, untersucht.

Lange wurde diese Forschung fast ausschließlich an der englischen Sprache durchgeführt, erst in den letzten Jahrzehnten werden auch stärker flektierende Sprachen, zu denen auch Deutsch zählt, untersucht.

Diebe
Dieb – plural: muss „e“ als Mehrzahlmarkierung erkennen

Tabelle 2-2 Beispiel Suffix -e als Pluralmarkierung (2)

Syntax:

Hier muss die Grammatikalität und darauffolgende die Bedeutung von Sätzen erkannt und verifiziert werden. Die Analyse der Struktur von Sätzen und Wörtern ist historisch einer der am stärksten ausgeprägten Teilbereich der Computerlinguistik.

Der gewitzte Dieb stahl das Geld.
Der Dieb gewitzt stahl das Geld. [ungrammatisch]

Tabelle 2-3 Beispiel Syntax Korrektheit (2)

Semmantik:

Hierbei wird die eigentliche Bedeutung der sprachlichen Einheit betrachtet.

Die Polizei beschlagnahmte das Diebesgut.
Das Diebesgut beschlagnahmte die Polizei.

Tabelle 2-4 Beispiel Semantik (2)

Die Veränderung der Satzstellung beeinflusst die inhaltliche Bedeutung des Satzes.

Pragmatik:

Dabei wird der Zweck einer sprachlichen Einheit hinterfragt.

Ist das Fenster offen?

Tabelle 2-5 Pragmatik Beispielsatz (2)

Was möchte der Autor damit bezwecken? Ist es schlicht eine Informationsfrage oder steht eine andere Motivation dahinter? Ist es zu kalt und die Frage ist als Aufforderung bzw. Bitte zum Schließen des Fensters zu interpretieren?

Man befasst sich mit der maschinellen Bestimmung des Antezedens, also einer impliziten Annahme resultierend aus einer Aussage.

Erkenntnisgewinne aus der Computerlinguistik haben aber auch Einfluss auf die Linguistik selbst.

Zu einer weiteren Aufgabe zählt die Evaluation linguistischer Theorien. Erst die Anwendung der Linguistiktheorien auf große Sprachdaten liefert Erkenntnisse zu deren Korrektheit und kann zu weiteren praktischen Verwendungen dieser Theorien führen.

Vor allem die Bearbeitung von großen Sprachkorpora ist erst durch die Verfügbarkeit von rechenleistungsstarken Computern möglich geworden und wird daher stark von der Informatik beeinflusst. Hier vor allem mit Wissen über Datenstrukturen und die effiziente Verarbeitung von großen Datenmengen, welche Milliarden an Wörtern umfassen können, aber auch theoretische

Aspekte über die Berechenbarkeit, Komplexitätstheorie, Aussagenlogik sowie Wissen über formale Sprachen.

Zur Spracherkennung, grammatischen Analysen und Semmantik werden Verfahren der Künstlichen Intelligenz genutzt. Programmiersprachen wie LISP und PROLOG werden genutzt. Semantische Beschreibung von Sprache basieren auf Prädikatenlogik.

Dieser sind in der Anwendung auf natürliche Sprache aber Grenzen gesetzt. Maschinelle Verarbeitung durch reine Aussagenlogik hat Probleme Verhältnismäßigkeiten im Sprachgebrauch zu erkennen, womit der Mensch keine Probleme hat wie folgendes Beispiel demonstriert:

Ein großer Berg.
Eine große Ameise.

Tabelle 2-6 Problem adäquater Beschreibung Beispiel (2)

In beiden Fällen ist mit dem Adjektiv „groß“ eine andere Skala zu verstehen.

Dasselbe Problem beobachtet man bei scheinbar widersprüchlichen Aussagen:

Vögel können fliegen.
Pinguine sind Vögel.
Pinguine können nicht fliegen.

Tabelle 2-7 Problem Aussagenlogik Beispiel (2)

Vögel können bekannterweise fliegen, Pinguine aber nicht. Um dies abbilden zu können werden Default-Mechanismen eingesetzt, die Standardannahmen bei gegensätzlichen Beweisen zurücknehmen.

Zur formalen Beschreibung natürlicher Sprache nutzt man Automatentheorie und formale Sprachen und entwickelt Berechnungsmodelle und Repräsentationsmechanismen.

Aus der Informatik ist weiters Graphentheorie nützlich in der Computerlinguistik. Viele Zusammenhänge lassen sich als Graphen, also aus Menge von Knoten, welche durch Kanten miteinander verbunden sind. In weiterer Folge werden wir in dieser Arbeit noch Baumbanken behandeln, welche Graphentheorie nutzen.

Aus der Mathematik spielt Statistik eine wichtige Rolle. Hierbei können in großen Sprachkorpora über statistische Aussagen die Verwendung von Sprache beschrieben werden. Automatische Erkennung von gesprochener Sprache wird fast ausschließlich über stochastische Automaten realisiert.

2.1.2 Anwendung

Computerlinguistik hat durch die starke Verbreitung von Computersystemen und deren omnipräsenz stark an Bedeutung für die Industrie gewonnen. Alle großen Websuchmaschinen nutzen erworbenes linguistisches Wissen. Jede Suchanfrage wird normalerweise einer morphologischen Analyse unterzogen, um nicht nur durch reinen Wort (Token) vergleich sondern tiefergehendem Verständnis Antworten zu liefern.

Metacrawler durchsuchen Webseiten und analysieren diese mit unüberwachten Clustering-Methoden und generieren Textzusammenfassungen, um z.B. aktuelle Nachrichtenlagen zu präsentieren.

Das Web enthält Millionen an Dokumenten, die oft aber nur schwer zugänglich sind und Sprachverarbeitung hilft bei der besseren Bereitstellung dieser.

Eine weitere Anwendung ist die Erstellung von Dialogsystemen. Diese ermöglichen einfachere Zugänge zu komplexen Systemen wie dem Bestellen von Flugtickets, etc. Diese können aber auch in Form von Chatbots durch ein Hilfesystem führen oder Reklamationen annehmen.

Übersetzungssysteme sind mittlerweile laufend anzutreffen und erlangen vor allem in einer globalisierten Welt große Bedeutung. Dies kann beim Übersetzen einer Bedienungsanleitung oder beim Schriftverkehr mit einem anderssprachigen Kollegen hilfreich sein.

2.2 TIGER Sprachkorpus

Der TIGER Sprachkorpus (TiGer treebank) wurde in mehrjähriger Zusammenarbeit des Computerlinguistik Instituts der Universität Saarbrücken, des Instituts für Maschinelle Sprachverarbeitung der Universität Stuttgart und des Instituts für Germanistik der Universität Potsdam entwickelt. Dieser Korpus umfasst in aktueller Version 2 ungefähr 900.000 Tokens (Wörter) bzw. 50.000 Sätze aus deutschen Zeitungsartikeln der Frankfurter Rundschau.

Diese Sätze wurden für den Korpus analysiert und alle Wörter wurden in Wortarten klassifiziert (POS-Tagging) und dabei mit Annotations (den Tags) sowie mit syntaktischen Informationen versehen, um diese maschinell verarbeiten zu können.

Dieser gesamte Korpus steht im NEGRA Export Format sowie im CoNNL-2009 Format auf der Webseite der Universität Stuttgart zur Verfügung und kann für nicht kommerzielle Forschungszwecke kostenlos verwendet werden. (3)

Das für diesen Korpus verwendete Annotationsschema um die grammatikalischen Informationen zu den Tokens abzubilden, orientiert sich am NEGRA Korpus (NEGRA treebank). (4) Hierbei wird der Korpus in Sätze geteilt und jeder Satz wird in seiner Struktur mit Annotationen beschrieben.

Diese Annotationen bestehen im Groben aus zwei Teilen: Informationen zu den einzelnen Wortarten, welche mit einem Tag zu jedem Wort oder zu jeder Wortgruppe abgebildet werden und syntaktische Informationen (im Speziellen die Satzstellung) die als Graph abgebildet werden. Dies ähnelt einer Baumstruktur (Kategorie als Knoten und Funktion als benannte Kanten) und deshalb werden diese Korpora auch als Treebank bezeichnet. Die Kanten die dabei von Knoten zu Knoten gehen beschreiben die Funktion der Kindknoten in Bezug auf deren Elternknoten. Zusätzlich gibt es auch sekundäre Kanten, welche Phrasen beschreiben können.

Der Fokus des TIGER Korpus liegt in der syntaktischen Annotation und in deren Klassifizierung von Token in Wortarten (POS). Dies passiert nach dem leicht modifizierten Stuttgart-Tubin Tagset (STTS) Annotationsschema.

Dieses Annotationsschema (Tagset) ist hierachisch strukturiert und umfasst elf Hauptwortarten sowie deren Unterwortarten, welche die Tags bilden. Diese Tags sind weitestgehend selbsterklärend und setzen sich aus Abkürzungen der Wortarten zusammen.
Folgende Tabelle umfasst die Hauptwortarten. Ein detailierteres Tagset mit Unterwortarten findet sich im Anhang dieser Arbeit.

Nomina	*N*	Adverbien	*ADV*
Verben	*V*	Konjunktionen	*KO*
Artikel	*ART*	Adpositionen	*AP*
Adjektive	*ADJ*	Interjektionen	*ITJ*
Pronomina	*P*	Partikeln	*PTK*
Kardinalzahlen	*CARD*		

Tabelle 2-8 Hauptwortarten des STTS Tagset

Nach der Klassifizierung in eine dieser Hauptwortarten erfolgt eine Unterteilung in weitere Kategorien, wobei jede Hauptwortart unterschiedlich stark unterteilt wird. Als Beispiel wird die Hauptgruppe Pronomina (Tag: P) in acht weitere Untergruppen unterteilt, welche wiederum unterteilt sein können. Diese richtet sich nach der Funktion der Pronomina und kann substituierend (Tag: S), attribuierend (Tag: AT) oder adverbial (Tag: AV) sein.

Das STTS Tagset besteht aus 54 Annotationen. 48 davon werden zum Wortklassifizieren (POS) verwendet und sechs Tags werden für fremdsprachliches Material (Tag: FM), Kompositions-Erstglieder (Tag: TRUNC), Nichtwörter (Tag: XY) und Satzzeichen (Tag: $,, $., $() verwendet. (5)

Im Tagset ist für jede Wortform genau ein Tag vorgesehen. Zu den Wortformen zählen hier auch Zahlen, Ziffern, Sonder- und Satzzeichen. Dies bedeutet, dass Sätze in der Regel an Leerzeichen getrennt werden, um die Wörter bzw. Wortformen einzeln zu taggen.

An folgendem Beispiel wird dies gezeigt:

Jetzt	solle	erneut	ein	Antrag	gestellt	werden	.
ADV	*VMFIN*	*ADJD*	*ART*	*NN*	*VVPP*	*VAINF*	*$*

Tabelle 2-9 POS Beispiel (6)

Sämtliche Annotationen werden in einer Baumbank (Treebank) gespeichert und können mit einer eigenen Abfragesprache (TIGER query language) gesucht werden. Die Ausdrücke dieser Abfragesprache beschreiben Knoten und Beziehungen zwischen Knoten und können flexibel mit boolschen Operatoren miteinander verknüpft werden.

Die einfachsten Knotenbeschreibungen werden als *feature constraint* bezeichnet, bilden ein *feature-value* Paar und können folgendermaßen aussehen:

[pos = „NN“]
[pos = („VVFIN“ | „VAFIN“ | „VMFIN“)]

Tabelle 2-10 Feature Constraint Beispiel (6)

Hier wird die POS Annotation abgefragt. Der erste Ausdruck liefert alle Nomen und der zweite Ausdruck liefert disjunkt alle Elemente, die mit einem der drei Tags versehene wurden.

Alternativ dazu können eigene Feature-Typen vordefiniert werden.

finite := VAFIN, VVFIN, VMFIN;
[pos = finite]

Tabelle 2-11 Feature Constraint Beispiel (6)

In diesem Beispiel wird *finite* als eigendefinierter Typ erstellt und filtert dann alle finiten Verben aus der Treebank heraus. Diese Typen können wiederbenutzt werden und sind Boolschen Ausdrücken vorzuziehen.

Die Syntax von Sätzen wird im Korpus als Graph abgebildet. Satzglieder werden über Knotenlabel beschrieben. Non-Terminale Label bezeichnen die Kategorie der Phrase und terminale Knoten stellen die Wortform mit dem POS Tag dar.

Folgendes Beispiel zeigt eine Nominalphrase:

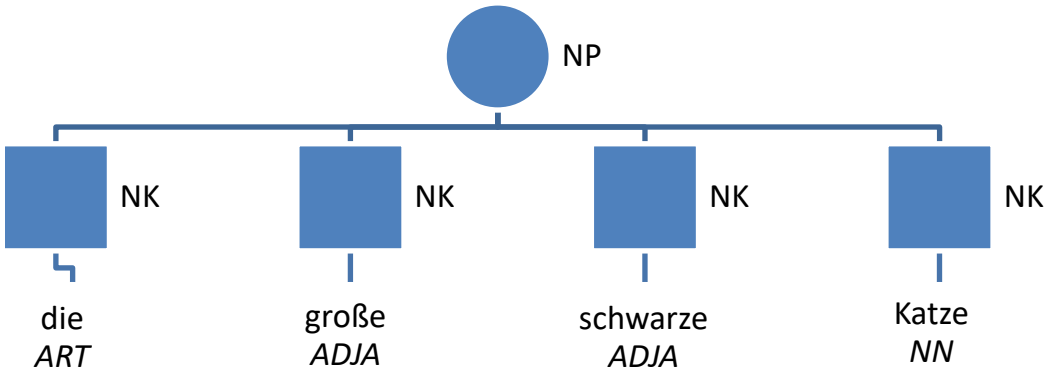
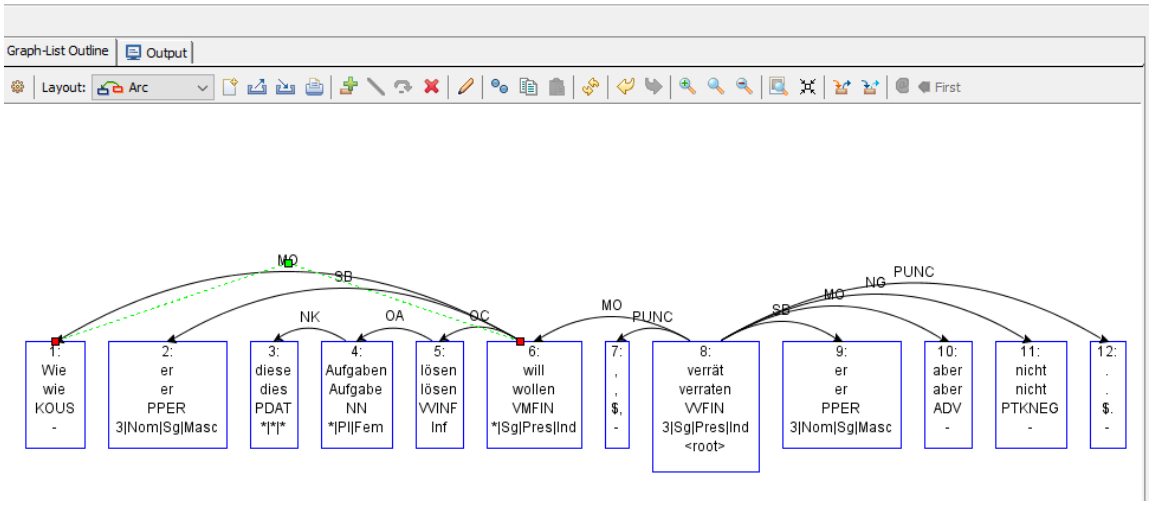


Tabelle 2-12 Nominalphrasen Beispiel (7)

NP … Nominalphrase

NK … Nomen Kernel, Teil der Phrase

Zum Abfragen und visuell darstellen der Graphen gibt es die Tools TIGERSearch und ICARUS (Interactive platform for Corpus Analysis and Research tools, University of Stuttgart) der Universität Stuttgart. ICARUS ist sehr flexibel und es können sehr viele im CoNLL 09 kodierte Korpora eingelesen werden.



Properties | Table Outline

#	Form	Lemma	Features	PoS	Head	Relation
1	Wie	wie		KOUS	6	MO
2	er	er	3|Nom|Sg|Masc	PPER	6	SB
3	diese	dies	*|*|*	PDAT	4	NK
4	Aufgaben	Aufgabe	*|Pl|Fem	NN	5	OA
5	lösen	lösen	Inf	VVINF	6	OC
6	will	wollen	*|Sg|Pres|Ind	VMFIN	8	MO
7	,	,		$,	8	PUNC
8	verrät	verraten	3|Sg|Pres|Ind	VVFIN	<root>	ROOT
9	er	er	3|Nom|Sg|Masc	PPER	8	SB
10	aber	aber		ADV	8	MO
11	nicht	nicht		PTK...	8	NG
12	.	.		$.	8	PUNC

Tabelle 2-13 ICARUS Treebank Manager Demo

2.4 Flache Satzverarbeitung

Um Texte korrigieren zu können, muss die Zeichenfolge von Maschinen verstanden werden. Dabei zerlegt man elektronische Dokumente in sprachlich relevante Einheiten. Dies können ganze Sätze oder Phrasen sein. Normalerweise bricht man diese aber in kleinere Teile herunter, teilt Wortfolgen in einzelne Wörter und interpretiert diese.

Wörter werden mit Taggern mit ihren Wortarten ausgezeichnet und danach von Chunk-Parsern zu Phrasen zusammengefasst. Text in Dokumenten, die mit solchen Annotationen versehen sind, bilden eine wertvolle Ressouce zu beliebiger Weiterverarbeitung.

Im Folgenden wird die Verarbeitung vom rohen Klartextdokument bis zur computerlinguistisch wertvollen Quelle für weitere Implementierungen beschrieben.

2.4.1 Tokenisierung

Vor der eigentlichen Verarbeitung von Texten trennt man diese in linguistische Einheiten. Meistens sind diese Einheiten Sätze oder Wörter.

Wort:

> „Ein Wort ist eine Einheit aus alphanumerischen Zeichen, die zu ihrer Rechten und Linken durch Leerraumzeichen (engl. *white space*) oder Interpunktion begrenzt wird.“ (2)

Diese Definition für ein Token trifft auf Schriftsysteme zu, die aus lateinischen, griechischen oder kyrillischen Schriftzeichen bestehen. Diese zählen zu den segmentierten Schriftsystemen.

Im Zusatz dazu gibt es nicht-segmentierte Schriftsysteme wie das Chinesische oder Japanische. In diesen Systemen fehlen meist Leerraumzeichen und Interpunktion und die Schriftzeichen (Piktogramme) werden nicht voneinander abgegrenzt sondern aneinander geschrieben. Schwierig hierbei ist, dass ein Piktogramm ein Wort aber auch eine Phrase abbilden kann und im Zusammenspiel mit benachbarten Piktogrammen die Bedeutung verändern kann. Nicht-segmentierte Sprachen werden hier nicht weiter behandelt.

Die Tokenisierung steht auch in segmentierten Schriftsystemen vor der Herausforderung, Token richtig zu erkennen. Man unterscheidet zwischen bedeutungstragenden Elementen wie Buchstaben, Ziffern und Satzzeichen und Zeichen, welche rein typographische Zwecke erfüllen. Zu den rein typographischen Elementen zählen Leerzeichen, Tabulatoren und Zeilenumbrüche.

Auf Mehrdeutigkeiten stößt man bei Satzzeichen wie dem Punkt (.). Dieser kann entweder ein Satzende markieren, aber auch eine Abkürzung anzeigen oder in Datums- und Zahlenangaben als Trennzeichen dienen. Hierbei muss man kulturelle bzw. regionale Unterschiede beachten.

Im englischsprachigen Raum werden bei Zahlen mit dem Punkt Dezimalstellen abgetrennt, im deutschsprachigen Raum werden damit aber Tausenderstellen in Zahlen angezeigt. In der Schweiz werden diese Tausenderstellen aber mit einem einfachen Hochkomma angezeigt. Die gleiche Zahl kann also je nach Sprachkontext anders dargestellt werden.

Die interessanteste Mehrdeutigkeit ist bestimmt der Punkt, wenn er entweder ein Satzende oder eine Abkürzung markiert. Bei einer Abkürzung gehört der Punkt nämlich zum Wort, aber beim Satzende ist dieser zu einem eigenen Token zu trennen. In beiden Fällen wird dieser aber einfach rechts zur Zeichenkette angehängt.

Hieaus ergibt sich, dass die eigentliche Herausforderung beim Tokenisieren das Erkennen von Satzende darstellt.

Allgemein werden Programme bzw. Algorithmen zur Tokenisierung von Texten in Symbolische und Statistische Tokenisierungsverfahren eingeteilt.

Bei symbolischen Verfahren entwickelt man eine Sammlung von Regulären Ausdrücken (Regular Expressions), um mit diesen Sätze und Wörter zu erkennen.

Bei statistischen Verfahren erstellt man lernende Systeme. Hierbei gibt es Anwendungen, die aus bestehenden Korpora lernen (Überwachtes Lernen), aber auch unüberwachte Systeme, die aus Regeln und Wahrscheinlichkeiten Satzgrenzen erkennen. Hierzu werden n-gramm-Modelle verwendet. Bei diesen betrachtet man n Wörter um ein potentielles Satzende herum, um mit dessen Vorkommenswahrscheienlichkeiten eine Entscheidung treffen zu können.

Part-of-Speech Tagging

Nachdem Texte in einzelne Sätze und Wörter aufgetrennt wurden, versucht man deren Wortarten zu erkennen. In solchen Systemen gibt es wieder überwachte und unüberwachte Verfahren. Überwachte Verfahren verwenden einen Annotierten Korpus und weisen einem eingelesenen Wort aufgrund von Vorkommenswahrscheinlichkeiten einen Tag zu. Unüberwachte Verfahren betrachten den Kontext eines Wortes und weisen diesem über bekannte Wahrscheinlichkeiten zu diesem Kontext ein Tag zu.

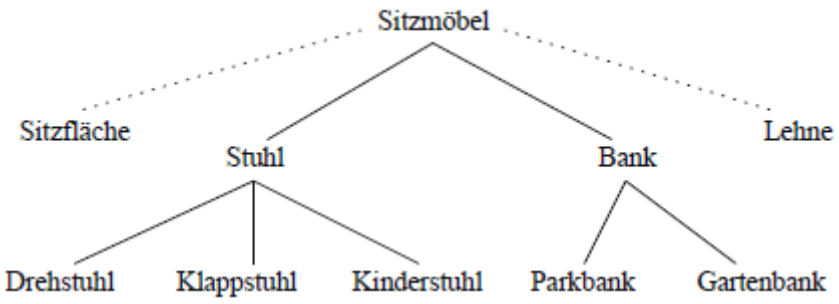
Tagger die mit Korpora trainiert wurden, werden zum Taggen meistens eingesetzt. Hierbei kann man rein durch nachsehen im Korpora sehr oft schon eine Trefferwahrscheinlichkeit von fast 90% erreichen und diese durch anschließende Transformationen noch erhöhen.

Ausschlaggebend für die Genauigkeit aber auch Geschwindigkeit eines Taggers ist das Tagset. Dieses beschreibt alle im Korpus vorkommenden Wortarten. Jede Wortart beschreibt dabei eine Wortform. Im Deutschen ist hierbei das Stuttgart-Tübingen-Tagset (STTS) das wohl bekannteste, welches in weiterer Folge verwendet und auch detailierter beschrieben wird.

2.5 GermaNet

GermaNet ist ein Netz, welches lexikalische Semantik abbildet. Dies nennt man auch Wortsemantik und ist ein Teilbereich der Linguistik. Dieses besteht aus Nomen, Verben und Adjektiven der deutschen Sprache und ist in lexikalische Einheiten gruppiert. Diese Einheiten drücken dieselben Konzepte aus und sind in sogenannte Synsets gruppiert, welche zueinander semantische Beziehungen definieren. Ein Synset ist eine Synonymie (Bedeutungsgleichheit). GermaNet funktioniert ähnlich wie WordNet für das Englische. Es kann als Thesaurus oder Ontologie verstanden werden. In einem Thesaurus findet man Beziehungen von Synonymen (Wörter mit gleicher oder sehr ähnlicher Bedeutung) zueinander. Als Ontologie versteht man in der Informatik eine Menge von Begrifflichkeiten und deren Beziehungen zueinander. (8)

In diesem Wortnetz ist ein Konzept nicht nur über seinen Knoten auffindbar, sondern auch über die Beziehung zu anderen. Ein Synset fasst nicht gleiche Wörter, sondern gleiche Bedeutungen zusammen. Somit unterscheidet man die Leseart eines Wortes und dies ist bei der maschinellen Übersetzung und dem semantischen Verständnis von großer Bedeutung. Folgend ein Beispiel für ein Konzept:



1 Semantisches Netz (2)

Hier sieht man Ober- und Unterbegriffe in Beziehung zueinander.

GermaNet wird an der Universität Tübingen vom Seminar für Sprachwissenschaft im Bereich der Allgemeinen Sprachwissenschaften und Computerlinguistik entwickelt und gewartet und wurde in das EuroWordNet (EWN) integriert, was eine multilinguale Sprachdatenbank ist.

Für dieses Projekt wurde eine Akademische Lizenz für die Forschungsstelle Österreichisches Deutsch ausgestellt. Diese befindet sich im Anhang.

In dieser Arbeit wurde die Version 10 von GermaNet verwendet. Diese Version umfasst folgende Inhalte:

101371	Synonymmengen
131814	Lexikalische Einheiten
119231	Literale
1.3	Lexikalische Einheiten pro Synonymmenge
114619	Anzahl konzeptueller Beziehungen
4199	Anzahl lexikalischer Beziehungen

Anhang 1 GermaNet Logo (8)

GermaNet wurde mit dem Ziel den deutschen Basiswortschatz abzudecken entwickelt und in den vergangenen Jahren laufend erweitert und ist mittlerweile sehr umfangreich. Es wurde aus verschiedensten Quellen wie dem Deutschen Wortschatz und dem Brockhaus manuell aufgebaut. Dabei wurden Korpusfrequenzen berücksichtigt. Damit ist gemeint, dass gebräuchlichere Wörter eher aufgenommen wurden. (2)

In GermaNet werden Lemmata verwendet. Dies ist die Grundform von Worten, in welchen diese üblicherweise in Wörterbüchern zu finden sind. Diese umfassen Nomen in Nominativ Singular (1. Fall Einzahl), Verben als Infinitiv und Adjektive geschlechtsneutral.

GermaNet besteht aus lexikalischen Einheiten, die in Synonymmengen (synsets) zusammengefasst werden. Diese Einheiten sind mit Beziehungen zueinander verbunden. Dabei unterscheidet man zwischen lexikalischen und konzeptuellen Beziehungen. Lexikalische Beziehungen sind die Synonymie und die Antonymie. Konzeptuelle Beziehungen beinhalten Hyponymie und Hyperonymie.

- Synonym:
  Intern in einem Synset bezeichnet dies die Wortgleichheit.

Folgende Bedeutungen können diese Synsets zueinander aufweisen:

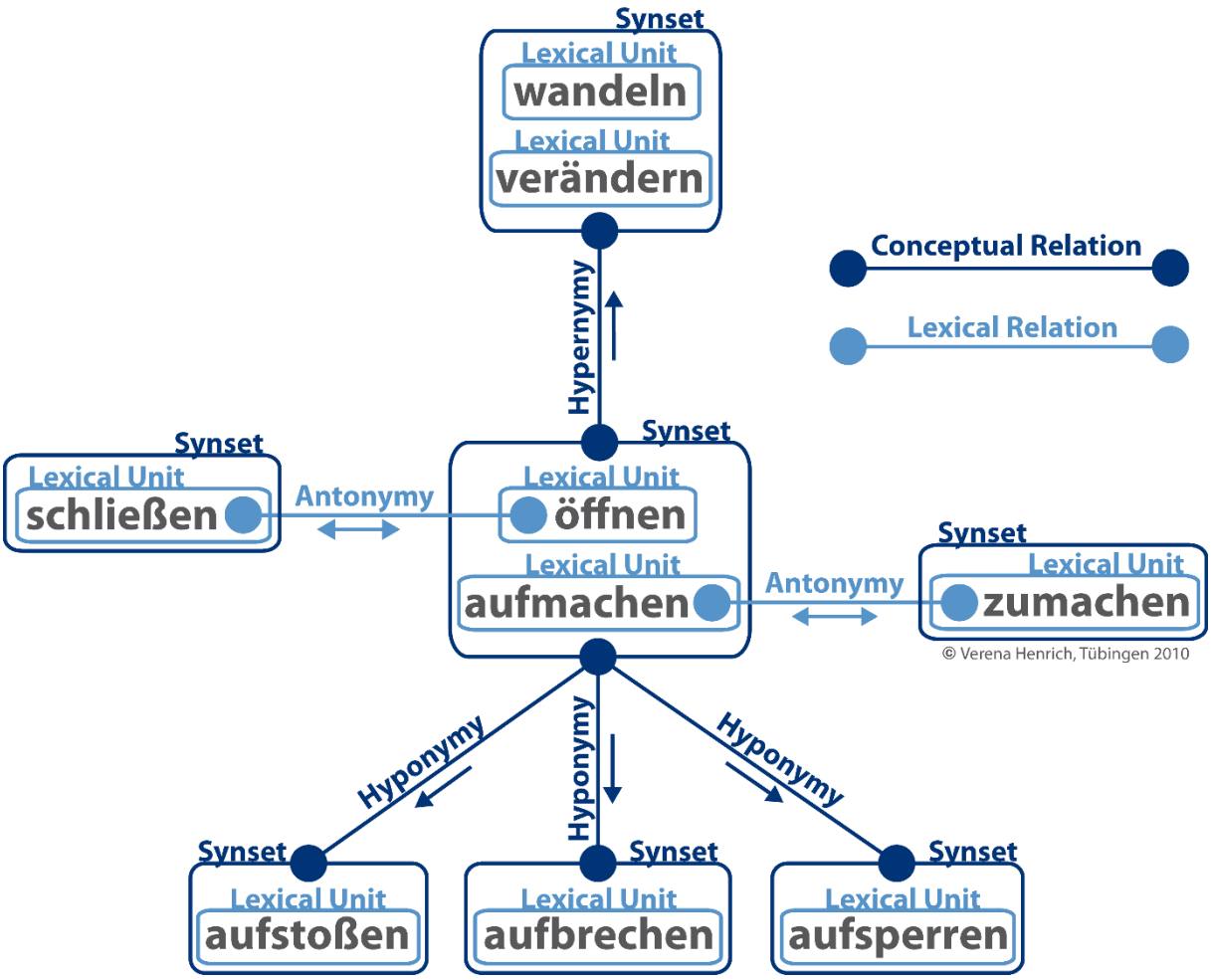
- Antonym:
  Unter Antonymen versteht man in der Linguistik Wörter mit Gegensätzlicher Bedeutung. Gegenteiligkeit.

- Hyponym:
  Hyponymie ist die Unterordnung von semantisch ähnlichen Begriffen. Man bezeichnet dies auch als Subordination und Hyponym wird auch Unterbegriff genannt.

- Hypernym:
  Dies ist das Gegenteil von Hyponym. Man nennt es auch Oberbegriff.

  (2)

Folgende Grafik beschreibt beispielhaft Synsets und die Beziehungen dieser zueinander:



2 GermaNet Beziehungen (8)

Ausgehend von den Synonymen in der Mitte *öffnen* ⇔ *aufmachen* gibt es lexikalische Beziehungen zu den Antonymen *zumachen* und *schließen*.

Zusätzlich gibt es von den Synonymen *öffnen* ⇔ *aufmachen* konzeptionelle Beziehungen zu drei Hyponyme *aufstoßen*, *aufbrechen* und *aufsperren*.

Als Hyperonyme zu *öffnen* ⇔ *aufsperren* findet man hier *wandeln* ⇔ *verändern*.

3 Webapplikation

3.1 Zielsetzung

Die grundsätzliche Anforderung an *oekorr* ist es eine Website zu schaffen, auf welcher Text eingegeben werden kann und automatisiert ein Vorschlag zur Ersetzung deutschländischer Wörter durch österreichischdeutsche unterbreitet wird.

3.2 Konzeption und Architektur

Um dieses Ziel zu erreichen wurde zuerst mit Prototypen zur Sprachverarbeitung experimentiert. Wenn man sich mit maschineller Sprachverarbeitung beschäftigt, stößt man auf viele Tools und Bibliotheken, die von Universitäten und Unternehmen angeboten werden. In den vergangenen Jahren wurde Sprachverarbeitung in praktischen Anwendungen deutlich vorangetrieben und ist durch Smartphones und Sprachsteuerungen immer präsenter. Die großen Cloudhoster entwickeln eigene Bibliotheken und Systeme, die man ansprechen kann. In dieser Arbeit werden freie Tools und keine Cloudlösungen verwendet. Schnell landet man damit bei Tools, welche in Python entwickelt wurden. Dies ist vor allem im universitären Umfeld anzutreffen. Somit fiel die Wahl der serverseitigen Programmiersprache auf Python und die verwendeten Technologien wurden dazu abgestimmt.

Es wurde anfänglich entschieden, dass *oekorr* als Webservice bereitgestellt werden soll. Dies ermöglicht es im Nachhinein, diese API mit nahezu allen Plattformen und relevanten Programmiersprachen verbinden zu können.

Das Webservice soll sehr schlank und performant sein. Nachdem einige Python Frameworks zur Webentwicklung verglichen wurden, fiel die serverseitige Entscheidung auf Flask. In dieses Framework werden die Skripte und Tools zur Sprachverarbeitung eingebaut. Dabei wurde großer Wert auf die ordentliche Verwendung von Webservicetechnologien gelegt. Es wird ein RESTful Webservice mit moderner, zustandsloser Authentifizierung genutzt. Die Daten kommen aus einer relationalen MySql und einer nicht-relationalen MongoDB, beides auf den Einsatzweck abgestimmt.

Die Schnittstelle soll clientseitig genutzt werden können. Ein REST Service kann direkt aus Javascript angesprochen werden, da es sich um einfache HTTP Anfragen handelt. Um eine moderne Webanwendung zu entwickeln, wurde ein Frontendframework genutzt und die Entscheidung fiel auf AngularJS. Dieses unterstützt MVVM im Browser und eine Zweiwegedatenbindung, welche es erlauben, große funktionale Webanwendungen zu entwickeln.

Mit diesem Überblick und der Entscheidungsfindung werden in diesem Kapitel einige Technologien näher eingeführt und beschrieben.

3.3 Servertechnologie

3.3.1 Webserver

Die Wahl des Webservers ist nebensächlich. Flask liefert für Entwicklungsumgebungen und kleine Projekte einen eigenen Webserver, kann aber auch sehr gut mit einem Apache oder nginx Webserver betrieben werden.

3.3.2 Webframework Flask

Flask ist ein leichtgewichtiges Webframework, das in Python entwickelt wurde und die WSGI (Web Server Gateway Interface) Schnittstelle zu Webservern verwendet. Es wurde 2010 vom Österreicher Armin Ronacher veröffentlicht und wurde seither neben dem Webframework Django zu einem der weitverbreitetsten in der Programmiersprache Python. Der Fokus von Flask liegt darin, möglichst keine Abhängigkeiten zu haben. Einzige Abhängigkeit ist die Templating-Bibliothek und die WSGI Bibliothek. Es ist durch Python Bibliotheken einfach erweiterbar. Die Basisfunktionalität ist minimal und es wird deshalb auch als Mikroframework bezeichnet. Es ist über Bibliotheken zu sehr vielen Datenbanksystemen kompatibel, erlaubt Object-Relational Mapping (ORM) und kümmert sich um Sessions innerhalb von Requests.

3 Flask Webframework Logo (9)

```python
from flask import Flask, escape, request

app = Flask(__name__)

@app.route('/')
def hello():
    name = request.args.get("name", "World")
    return f'Hello, {escape(name)}!'
```

```
$ env FLASK_APP=hello.py flask run
 * Serving Flask app "hello"
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

4 Flask Mini-Beispiel (10)

3.4 REST Basics

Nach der Entstehung des Webs in den 90ern gab es bald eine Skalierungskrise, da HTTP1 und URL (Uniform Ressource Locator) Schwächen aufwiesen. Mit dem Hypertext Transfer Protocol HTTP1.1 wurde die Syntax von URI (Uniform Resource Identifier) standardisiert und das Web konnte weiter skalieren. Im Jahr 2000 hat Roy Fielding in seiner Dissertation einen standardisierten Stil für die Architektur des Webs beschrieben. Hierbei geht es vor allem um die semantisch richtige Verwendung der HTTP Methoden.

Webservices werden als APIs (Application Programming Interfaces) bezeichnet, um es Anwendungen zu ermöglichen, miteinander über das Web zu kommunizieren. Eine API stellt hierbei eine Menge an Daten und Funktionen bereit, um diese zu nutzen und zu verändern. Darüber können Computer Daten miteinander austauschen. Web APIs werden damit als das Gesicht des Webservices dargestellt, das auf Anfragen reagiert.

Daraus entstand der REST Architekturstil und nur Webservices, die sich daran halten, können sich als RESTful bezeichnen.

5 Web API (11)

Gut gestaltete Web APIs erhöhen die Attraktivität eines Webservice.

Hierfür wurden REST API Designregeln festgelegt und es wurde ein Framework zum Beschreiben von Web APIs entworfen, welches sich Web Resource Modeling Language (WRML) nennt.

Diese Regeln werden hier kurz zusammengefasst genannt.

- URI Format:
  URI = scheme "://" authority "/" path ["?" query] ["#" fragment]
- URI Authority Design:
  Hier werden Namenskonventionen beschrieben-
- Resource Modeling:
  Im URI soll mit jedem Slash (/) ein weiteres Segment des Datenmodells erreicht werden.
- Resource Archetypes:
  Es werden vier Archetypen genannt. Document (Entität in einer Datenbank), Collection (eine Art Verzeichnis am Server), Store (clientseitige Verwaltung von Items), Controller (Ein- und Ausgabe, Methoden)
- URI Path Design:
  Sinnvolle Benennung von Teilen zwischen der Slashes
- URI Query Design

Weiters wird die Verwendung der HTTP Methoden beschrieben. Web APIs sollen CRUD (Create, Read, Update, Delete) unterstützen. Folgende Tabelle gibt eine Übersicht zu deren Verwendung.

Delete	http Request um ein Element zu löschen
Get	Ruft Daten ab
Post	Damit werden Daten neu hinzugefügt oder eine Controllermethode aufgerufen
Put	Damit werden Daten aktualisiert
Head	Ruft Metadaten ab zum Zustand von Ressourcen
Options	Ruft die verfügbaren Methodeninformationen ab

Tabelle 3-1 Web API HTTP Methoden (11)

Nachfolgend eine Auflistung der Antwort Status Codes einer Web API.

1xx	Informationen des Protokolls zum Transfer
2xx	Erfolgsbenachrichtigungen
3xx	Information über eine Weiterleitung. Client muss reagieren.
4xx	Ein Fehler auf Clientseite
5xx	Ein Fehler auf Serverseite

Tabelle 3-2 HTTP Status Codes (11)

3.5 Webservice Authentifizierung

Der Zugang zu gewissen Webservice Methoden soll geschützt werden, damit diese nur von berechtigten Usern verwendet werden können. Da HTTP ein zustandsloses Protokoll ist, startet jeder Request wieder informationslos und muss auch Informationen zur Authentifizierung mitliefern. Dazu gibt es verschiedene Lösungen und Varianten. In der einfachsten Form kommt *HTTP-Authentifizierung* (12) zum Einsatz. Wie in RFC 2617 beschrieben umfasst diese die *Basic* und die *Digest Access Authentication*. Bei dieser Variante werden Benutzername und Kennwort bei jedem Request im http Header mitgesendet und vom Service überprüft. Sollte dies nicht der Fall sein, Antwortet das Service mit dem Statuscode *401 Unauthorized*.

Dies ist zwar einfach zu implementieren, aber dazu müssen die Zugangsdaten immer auf allen Client Anwendungen vorhanden sein. In modernen Servicearchitekturen ist dies unpraktikabel und stellt ein Sicherheitsrisiko dar. Weiters werden große Anwendungen auf viele Microservices aufgeteilt, an welchen sich Clients authentifizieren müssen. Dies bedeutet in weiterer Folge, dass jedes Microservice die Zugangsdaten validieren muss. Um diese validieren zu können, muss das Service auch Zugriff auf diese haben. In Servicearchitekturen muss man nun alle Benutzerdaten auf alle Services verteilen oder jedes Service muss zentral auf Authentifizierungsdaten zugreifen. Dies führt zu verzögerter Aktualität der Zugangsdaten bzw. stellt das zentrale validieren einen Flaschenhals dar.

Eine Lösung dieser Probleme ist die Verwendung einer tokenbasierten Authentifizierungsmethode. Seit 2015 existiert der RFC 7519 (13) Standard zu *JSON Web Token*. Dies ist eine moderne und mittlerweile weit verbreitete Norm für Zugangstokens. JWT wurde in den Webservices und der SPA Anwendung von *oekorr* verwendet und wird mit den klassischen Methoden nachfolgend beschrieben.

3.5.1 Basic Authentication

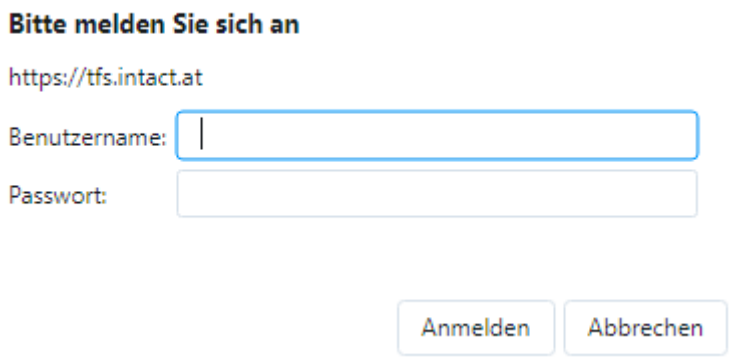
Diese Variante wird häufig verwendet und ist einfach zu implementieren. Durch den Statuscode 401 und dem *WWW-Authenticate* Feld in der Antwort erkennt ein Webbrowser, dass es sich um einen geschützten Bereich handelt und reagiert mit einem Prompt zur Eingabe von Benutzernamen und Kennwort. In einem Webservice muss der Konsument darauf reagieren bzw. den Aufruf einfach scheitern lassen, da er nicht den Voraussetzungen entspricht.

Die Authentifizierungsdaten werden bei dieser Methode im HTTP Header im Feld *Authorization* übertragen. Dazu werden Benutzername und Kennwort mit einem Doppelpunkt getrennt *Base64* codiert übertragen. Da dies lediglich codiert und nicht verschlüsselt ist, können diese Zugangsdaten mitgelesen werden. Dies ist einer Übertragung im Klartext gleichzusetzen und entspricht keineswegs den üblichen Sicherheitsansprüchen von Anwendungssoftware. Um Basic Auth trotzdem nutzen zu können, muss man die HTTP Verbindung mit SSL/TLS verschlüsseln. Dabei wird die Verbindung bereits vor der Übertragung gesichert und der Klartext im HTTP Header Feld stellt kein Risiko mehr dar. Weiters kann man sich damit auch vor *man-in-the-middle* Attacken bzw. gefälschten Empfängerservern schützen, da für eine HTTPS Verbindung ein gültiges Zertifikat erforderlich ist und dieses vom Sender validiert werden kann.

Beispiel einer Authentifizierungsanforderung des Webservers:

```
WWW-Authenticate: Basic realm="Mein Webserver"
```

Realm kann als Text angegeben werden und benennt den geschützten Bereich, für welchen man sich anmeldet.



6 Realm Prompt in Opera auf Windows

Beispiel einer Übermittlung an den Webserver / das Webservice:

```
Authorization: Basic bWF4aW1pbGlhbjpwYXNzd29ydA==
```

Base64 kodiert in der Form *Benutzername:Kennwort*. Hier: *maximilian:passwort*

3.5.2 Digest Access Authentication

Diese Methode erweitert das oben beschriebene Basic Auth. Bei der Aufforderung zur Eingabe von Benutzername und Kennwort sendet der Webserver im Authenticate Header auch noch ein Nounce mit. Dieses wird vom Client gemeinsam mit dem Benutzernamen, Kennwort, URL und Protokoll an eine Hashing Funktion übergeben und dadurch ein String berechnet. Dieser String wird dann im HTTP Header an den Webserver gesendet und dieser berechnet auch eine Prüfsumme und kann den Benutzer somit authentifizieren.

Hier wird das Kennwort nun nicht mehr im Klartext übertragen und solange es sich um eine starke Hashingfunktion handelt, kann dieses auch nicht reproduziert bzw. zurückgerechnet werden. Die Übermittlung des Kennworts ist somit sicher.

Da hierfür aber ein Ping-Pong notwendig ist, wird diese Methode in Webservices kaum eingesetzt

3.5.3 JSON Web Token

Die Kurzfassung der RFC 7519 beschreibt JWT folgendermaßen:

„JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.“ (13)

Bei JWT handelt es sich um einen offenen Standard, der eine schlanke und eigenständige Möglichkeit beschreibt, um Informationen sicher, als JSON Objekt oder String zwischen zwei Parteien zu übertragen. Diese übertragenen Informationen können von den Parteien bzw. Endstellen verifiziert werden und sind dadurch vertrauenswürdig. Dies wird durch digitales signieren der Informationen mit einem HMAC (Hash-Based Message Authentication Code) Algorithmus oder einer asymmetrischen RSA Verschlüsselung (public/private Key) erreicht.

Unter schlank (compact) versteht man die geringe Größe (wenigen Zeichen) in Bytes, die ein JWT benötigt. Man kann diese über http GET (im URL), HTTP POST oder auch im HTTP Header (als Bearer) benutzen bzw. mitsenden. Das erlaubt eine schnelle Übertragung und ist somit auch für leichtgewichtige Anwendungen benutzbar.

Mit eigenständig erfüllt ein JWT den Gedanken einer zustandlosen Verbindung und liefert alle relevanten Informationen direkt im Token mit. Dadurch ist es nicht weiter nötig, Informationen des Benutzers von einem Authentifizierungsservice (Identity Provider) zu holen, sondern kann direkt aus dem Token ausgelesen werden. Das erspart in Microserviceumgebungen zusätzliche Vernetzungen von Services.

Dies führt uns zu den möglichen Einsatzgebieten von JWT.

Informationsaustausch

Wie bereits beschrieben ist JWT eine gute Variante, um Informationen sicher, signiert zu übertragen. Durch die Nutzung asymmetrischer Verschlüsselung kann auch die Authentizität des Senders verifiziert werden. Dabei wird das JWT mit dem privaten Schlüssel signiert oder verschlüsselt und mit dem öffentlichen Schlüssel kann dies entschlüsselt bzw. die Signierung geprüft werden. Damit handelt es

sich definitiv um den korrekten Absender. Da Kopfdaten und Transportdaten gemeinsam zur Berechnung der Signatur verwendet werden, kann das JWT nicht manipuliert werden.

Authentifizierung:
Beim Einloggen eines Benutzers wird ein JWT generiert und dieses wird von nun an bei jedem Request mitgesendet. Damit wird es dem Benutzer ermöglicht, sich an Webseiten, Webservices und anderen Ressourcen anzumelden. JWT ist sehr gut für Single-Sign-On Systeme und Szenarios geeignet, da es neben des geringen Datentransfers auch einfach zwischen unterschiedlichen Domains genutzt werden kann.

Mittlerweile wurde JWT in fast alle großen Webframeworks implementiert und wird auf Grund der Kompaktheit und einfachen Handhabung vielen anderen, komplexeren Systemen vorgezogen. (14)

In folgenden Anwendungsbereichen findet man JWTs in Verwendung:

- Authentifizierung
- Autorisierung
- Föderierte Identitäten
- Clientseitige Sessions (zustandslose Sessions)
- Clientseitige geheime Informationsspeicherung

Struktur und Funktionsweise

Ein JWT Token besteht aus drei Teilen.

- Header
- Payload
- Signature

Diese werden durch Punkte getrennt.

hhhhhh.pppppp.ssssss

Tabelle 3-3 JWT Token Struktur

Das gesamte JWT ist ein gültiges JSON und kann zu einem Javascript Objekt deserialisiert werden.

Der Header besteht aus zwei Eigenschaften. Dem Typ des Objekts, also „JWT“ und dem Hashing Algorithmus. Dies könnte HMAC SHA256 oder RSA sein.

Die Payload des Tokens enthält die Informationen. Diese werden als Claims bezeichnet. Deren Bezeichnungen sind nur drei Zeichen lang. Hierbei unterscheidet man zwischen drei Typen.

Reservierte Claims werden vorgegeben und es wird empfohlen, diese zu nutzen. Beispiele sind exp (expiration time), sub (subject) oder iss (issuer).

Öffentliche Claims können frei gewählt werden. Man kann diese aber bei der IANA JSON Web Token Registry hinterlegen.

Private Claims können frei gewählt werden und werden nirgends registriert.

Die Signatur des Tokens wird aus dem Base64 kodierten Header sowie die Base64 kodierte Payload und einem Geheimnis (Secret) berechnet. Dazu wird ein Hashingalgorithmus verwendet. Mit der Signatur kann später der Ersteller überprüft werden und die Nachricht wird damit vor Manipulation geschützt.

Folgendes Beispiel zeigt die Erstellung eines Tokens und wurde auf *jwt.io* berechnet:

```
{
"alg": "HS256",
"typ": "JWT"
}
```

7 Header

```
{
  "sub": "0730425",
  "name": "Max Schafzahl",
  "iat": 1516239022
}
```

8 Payload

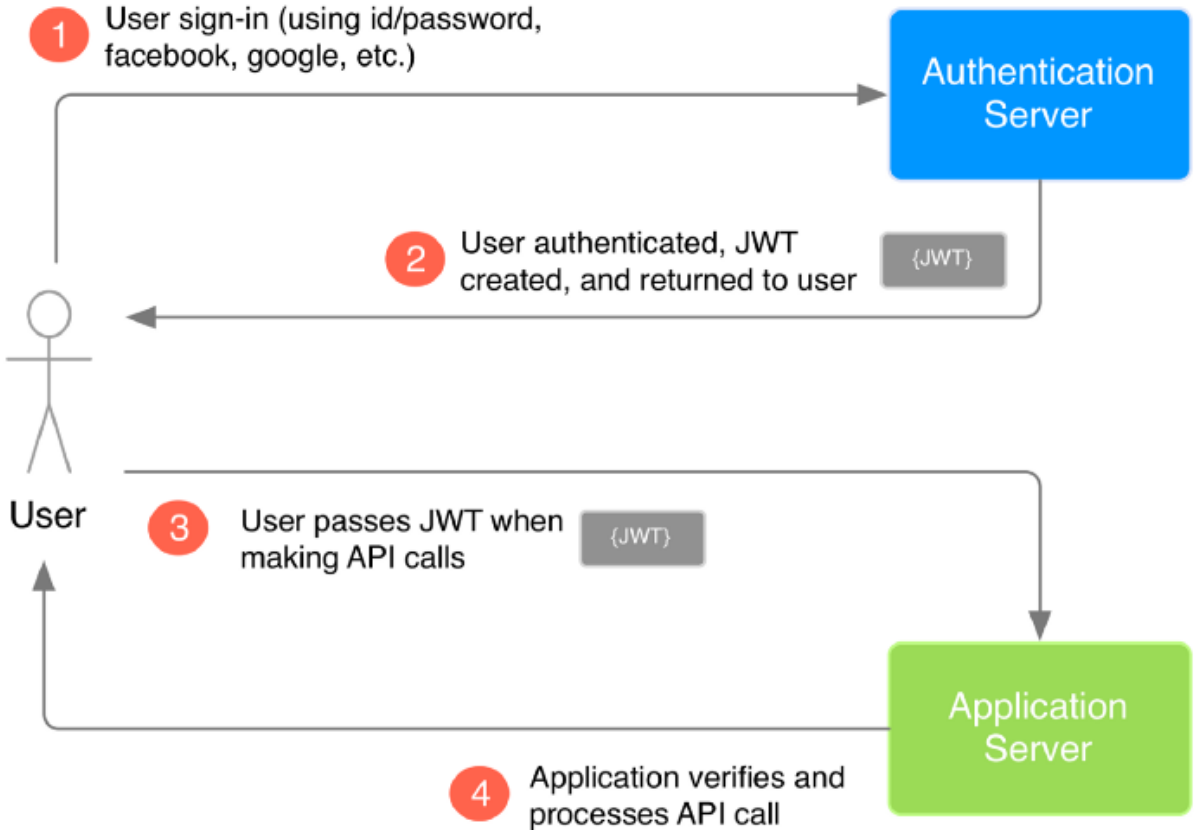
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),

your-256-bit-secret
)
```

9 Signatur Berechnung

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWI
iOiIwNzMwNDI1IiwibmFtZSI6Ik1heCBTY2hhZnphaGw
iLCJpYXQiOjE1MTYyMzkwMjJ9.rsjtcejSb274NzTBuG
a2BtB5MvAIi-KfX65Vqa4Ia8c
```

10 Berechnetes JWT

Nachfolgende Grafik skizziert den Ablauf einer Authentifizierung an einem System mit der JSON Web Token Technologie.



11 Authentifizierungsflow bei JWT (15)

Hierbei loggt sich der Benutzer erfolgreich mit seinen Zugangsdaten an einem Authentifizierungsserver ein. Dieser sendet dann ein JWT an den Benutzer. Dieses muss lokal abgespeichert werden. In Browsern nutzt man dafür die Local Storage, Session Storage oder ein Cookie, wobei die Local Storage meist bevorzugt wird, da diese auch nach beenden des Browsertabs noch verfügbar und von allen Tabs zugänglich ist.

In jedem HTTP Request wird dieses dann im Header mitgesendet. Hierzu nutzt man das Authorization Feld und schreibt das Wort Bearer vor das Token. Dies ist ein zustandsloser Mechanismus und wird vom Server validiert aber nicht abgespeichert. Es muss also bei jedem Request wieder mitgesendet werden.

Bei Cookies gilt die Same-Origin Policy (16). Dieser Sicherheitsmechanismus bedeutet, dass nur Skripte und Dokumente derselben Quelle miteinander agieren dürfen. Damit wird potentieller schadhafter Zugriff auf andere Quellen verhindert. Dies bedeutet für Cookies, dass diese in Protokoll, Domain (Sublevel, Domainname und Toplevel) und Port übereinstimmen müssen. Es ist also nicht möglich, ein

Cookie mit einem Token an mehrere API Endpunkte unterschiedlicher URIs zu senden. Bei modernen Webservices benötigt man aber APIs an unterschiedlichen Endpunkten. Wenn man JWTs als Parameter oder HTTP Header Feld mitsendet, ist dies aber möglich. Man nennt dieses Anwendungsgebiet *Cross-Origin Resource Sharing (CORS)*.

Jedes JWT hat ein maximales Gültigkeitsdatum und kann nach Ablauf nicht mehr verwendet werden. Man ein gültiges JWT verwenden, um ein neues verlängertes zu bekommen.

3.6 Sprachverarbeitung mit NLTK

Das Natural Language Toolkit (NLTK) wurde 2001 im Rahmen eines Computerlinguistiklehrveranstaltung eines Informatikinstituts der Universität Pennsylvania (USA) gestartet und wird seither von sehr vielen Universitäten und Personen weltweit weiterentwickelt. Vier primäre Ziele wurden bei der Entwicklung beachtet:

- Einfachheit
  Dieses Framework soll sehr einfach zu verwenden sein, ohne dass die Entwickler tiefgründiges Wissen über Sprachverarbeitung besitzen. Mit diesem Toolkit soll es ermöglicht werden, praktisches, anwendungsorientiertes Wissen über NLP zu erlangen.
- Konsistenz
  Einheitliche Datenstrukturen und Schnittstellen sollen bereitgestellt werden. Alle Teile des Frameworks sollen einheitlich konsistent zu benutzen sein.
- Erweiterbarkeit
  Neue Softwaremodule sollen einfach dazu entwickelt und eingebunden werden können. Alle Namespaces und Interfaces sind darauf ausgerichtet.
- Modularität
  Jedes Modul soll eigenständig benutzbar sein, ohne den Rest des Frameworks zu kennen und zu beherrschen.

Folgende Module werden bereitgestellt:

NLTK Modul	**Einsatzgebiet, Funktionalität**
nltk.corpus	Zugriff auf Corpora
nltk.tokenize, nltk.stem	Textverarbeitung, Tokenizer für Sätze und Stemmer
nltk.collations	Collocations, Phrasen, t-tests, etc.
nltk.tag	Part-of-Speech Tagging
nltk.classify, nltk.cluster	Klassifizierung; Decision trees, naive bayes, k-means

nltk.chunk	Chunking, Bündeln; regex, n-gram
nltk.parse	Parsing
nltk.sem, nltk.inference	Semantische Interpretation; Modelchecking
nltk.metrics	Kennzahlen, Metriken berechnen; precision
nltk.probability	Wahrscheinlichkeiten, Schätzungen; Verteilungen von Termen
nltk.app, nltk.chat	Anwendungen; z.B. WordNet browser, chatbots
nltk.toolbox	SIL Toolbox

Tabelle 3-4 NLTK Module (17)

Dieses Toolkit ist kein allumfassendes System zur Sprachverarbeitung, sondern flexibel erweiterbar. Es soll aber trotzdem für praktisch relevante Einsätze genutzt werden. Es wurde nicht auf die Ausführungsgeschwindigkeit und Performance höchsten Wert gelegt, sondern auf eine einfache und plattformunabhängige Verfügbarkeit. Viele Algorithmen könnten in hardwarenahen Programmiersprachen wie C++ effizienter und schneller ausgeführt werden. Hier fiel die Entscheidung aber auf die einfache, aber mächtige Sprache Python und dessen Interpreter. Python eignet sich sehr gut zur Verarbeitung von Sprachdaten und steht gratis und plattformunabhängig zur Verfügung. Python wird interpretiert, hat eine einfache Syntax und ist objektorientiert. Sie eignet sich sowohl zur Entwicklung einfacher Skripts als auch großer Anwendungen. UTF8 wird voll unterstützt und ermöglicht das Verarbeiten aller Sprachzeichen. Listen und Collections von String lassen sich in Python sehr gut verarbeiten.

(17)

Für *oekorr* ist vor allem maschinelle Übersetzung interessant. Machine Translation (MT) ist eines der wichtigsten Einsatzgebiete von natürlicher Sprachverarbeitung (NLP). Seit Jahrzehnten versucht man Texte vollautomatisch zu übersetzen und profitiert von großartigen NLP Technologien, stößt dabei aber auch an Grenzen. Übersetzungen existieren in der Praxis für Sprachpaare. Deutsche Texte kann man automatisiert ins Englische übersetzen lassen. Um deutsche Texte aber ins Spanische zu übersetzen, nimmt man den Umweg über eine gemeinsame Sprache. Das bedeutet, dass man viele Sprachen in Englisch übersetzen kann und dann von Englisch weiter in eine Zielsprache. Dabei entstehen zusätzliche Fehler, man muss aber nicht jedes Sprachpaar extra unterstützen.

Mit Babelizern kann man Texte in Schleifen in einem Sprachpaar alternierend übersetzen lassen und sieht, wie sich die Bedeutung verändert. Hier ein Beispiel:

```
>>> babelize_shell()
Babel> The pig that John found looked happy
Babel> german
Babel> run
0> The pig that John found looked happy
1> Das Schwein, das John fand, schaute gl?cklich
2> The pig, which found John, looked happy
```

12 NLTK Babelize (17)

3.6.1 Corpora

In Kapitel 2.2 wurde der TIGER Sprachkorpus für die deutsche Sprache vorgestellt. In NLP Systemen ist die Verarbeitung großer Sammlungen von linguistischen Daten notwendig. Das NLTK ermöglicht das Einlesen dieser Korpora. Ein Korpus ist eine große Menge Text, die in der Regel mit zusätzlichen Metainformationen versehen sind. Die Texte werden von homogenen Quellen gewählt. Dies kann eine Sammlung von Artikeln einer Zeitung oder eine Sammlung von Reden und Ansprachen sein.

Das NLTK bietet Zugriff auf viele Korpora, Grammatiken und trainierten Modellen. Mit dem NLTK Data Tool kann man diese herunterladen und installieren. NLTK bietet Methoden zum Einlesen und Nutzen dieser.

Eine der größten Sammlungen von Texten für das NLTK ist das Projekt Gutenberg. Hier werden mehr als 59000 eBooks gratis zum Download angeboten. Mit folgenden Anweisungen kann man die lokalen Dateien auflisten:

import nltk
nltk.corpus.gutenberg.fileids()

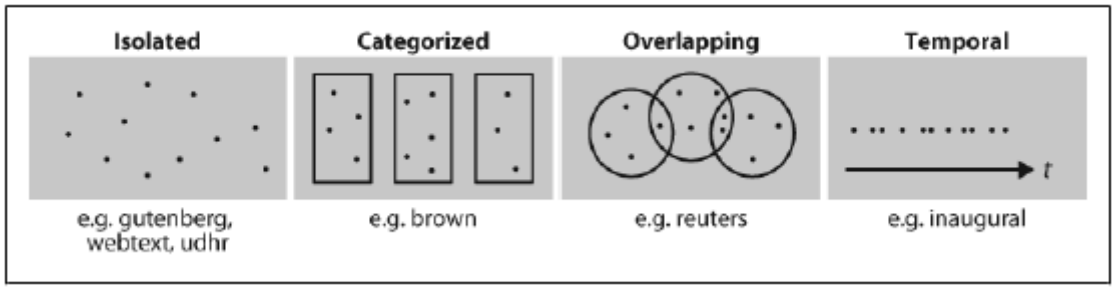
Bei Projekt Gutenberg handelt es sich um eine reine Textsammlung. Viele Textkorpora enthalten aber auch linguistische Anmerkungen. Diese nennt man *Annotated Text Corpora*. Ein großer Korpus ist der Brown Corpus. Dies war der erste englischsprachige Korpus mit mehr als einer Million Worten. Er wird in viele Kategorien wie Nachrichten, Hobbys, etc. unterteilt.

Bei diesen linguistischen Anmerkungen handelt es sich meistens um Part-of-Speech Tags. Das bedeutet, dass die Wortarten bestimmt und markiert wurden. Neben dem Brown Corpus können hier noch der CESS Treebanks, CoNLL 2000 Chunking Data und WordNet genannt werden.

Die meisten Korpora sind in Englisch. Für Deutsch ist der TIGER Korpus am bedeutendsten.

Die Struktur der Korpora kann systematisch sein. In der einfachsten Variante ist es einfach eine Sammlung von Texten. Diese Texte können gruppiert sein und diese Gruppen können sich überschneiden, wenn Texte zu mehreren Gruppen passen.

Folgende Grafik zeigt dies schematisch:

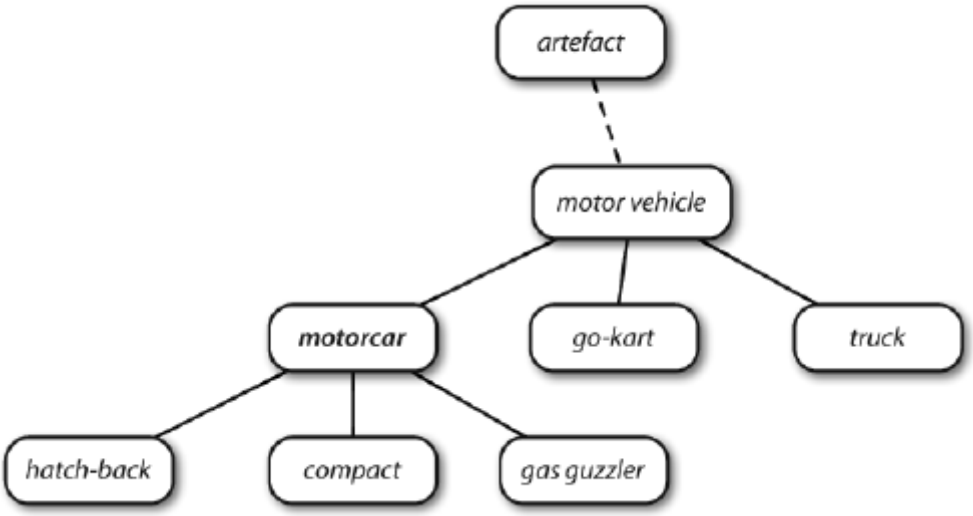


13 Strukturen von Textkorpora (17)

- Isoliert
- Kategorisiert
- Überschneidend
- Zeitlicher Verlauf

Einer der bekanntesten und wertvollsten Korpora ist das WordNet. Dieses Netzwerk besteht aus Synsets und deren Beziehungen zu einander. Dieses wurde für die englische Sprache entworfen. Das in einem vorhergehenden Kapitel beschriebene GermaNet orientiert sich an WordNet. Diese Netze funktionieren analog zu einander.

Nachfolgend ein Beispiel zu dessen Verwendung im NLTK:



14 Konzept Beispiel aus WordNet (17)

Mit WordNet und NLTK kann man sehr einfach zwischen den Konzepten navigieren. Hier wird das Synset des *Motorcar* genommen und davon werden alle Hyponyme ausgelesen.

```
>>> motorcar = wn.synset('car.n.01')
>>> types_of_motorcar = motorcar.hyponyms()
>>> types_of_motorcar[26]
Synset('ambulance.n.01')
>>> sorted([lemma.name for synset in types_of_motorcar for lemma in synset.lemmas])
['Model_T', 'S.U.V.', 'SUV', 'Stanley_Steamer', 'ambulance', 'beach_waggon',
'beach_wagon', 'bus', 'cab', 'compact', 'compact_car', 'convertible',
'coupe', 'cruiser', 'electric', 'electric_automobile', 'electric_car',
'estate_car', 'gas_guzzler', 'hack', 'hardtop', 'hatchback', 'heap',
'horseless_carriage', 'hot-rod', 'hot_rod', 'jalopy', 'jeep', 'landrover',
'limo', 'limousine', 'loaner', 'minicar', 'minivan', 'pace_car', 'patrol_car',
'phaeton', 'police_car', 'police_cruiser', 'prowl_car', 'race_car', 'racer',
'racing_car', 'roadster', 'runabout', 'saloon', 'secondhand_car', 'sedan',
'sport_car', 'sport_utility', 'sport_utility_vehicle', 'sports_car', 'squad_car',
'station_waggon', 'station_wagon', 'stock_car', 'subcompact', 'subcompact_car',
'taxi', 'taxicab', 'tourer', 'touring_car', 'two-seater', 'used-car', 'waggon',
'wagon']
```

15 NLTK WordNet Beispiel (17)

4 Single Page Application

4.1 Prinzip und Einsatz

In den Anfangsjahren des Web wurden Inhalte fast ausschließlich als statische Dokumente vom Webserver an den Webbrowser übertragen und dort dargestellt. Sämtliche Dokumente waren fertig erstellt und wurden unverändert im Webbrowser angezeigt. Diese sind relativ einfach mit HTML (Hypertext Markup Language) als Markup Sprache zu erstellen und zu interpretieren. Diese erlaubt das Layoutieren und Strukturieren von Texten und Bildern und das Referenzieren auf weitere Dokumente über Hyperlinks. Sämtliche Dokumente können mit Metadaten versehen werden, um zusätzliche Informationen für Maschinen lesbar zu transportieren.

Diese statischen Dokumente haben den Nachteil, dass man jede Änderung manuell vornehmen muss bzw. Informationen nicht dynamisch zum Abrufzeitpunkt getauscht werden können. Um dies zu ermöglichen, wurden serverseitige Skriptsprachen entwickelt, die zur Anfragezeit erst das HTML Dokument generieren. Weitverbreitete Programmiersprachen hierfür sind PHP, Python, Java und C#. PHP und Python werden interpretiert und können einfach in HTML Dokumente eingebettet werden. Um nun aber nicht nur einfache Webdokumente sondern große Webanwendungen erstellen zu können, wurden serverseitige Webframeworks entwickelt. Diese geben wartbare Strukturen vor und übernehmen sehr viel grundlegende (Boilerplate) Arbeit wie die Handhabung von Sessions, Zugriffe auf Datenbanken oder das Generieren bzw. Rendern von HTML-Controls.

Bekannte Frameworks sind CakePHP, Flask für Python, JavaServer Faces oder ASP.Net Webforms. Diese Frameworks erlauben es mit ihren Render-Engines (HTML-Rendering) Formulare und Seiten rein serverseitig zu definieren und programmieren und generieren dann das notwendige HTML daraus, um es als GUI im Browser darzustellen.

Rein serverseitig gerenderte Webanwendungen erfordern zum Navigieren und Abschicken von Formulardaten einen vollen GET oder POST HTTP Request. Dabei werden die POST Daten an den Server geschickt, verarbeitet und danach das neu generierte HTML Dokument an den Browser übertragen. Das Browser *document* wird dabei komplett entladen und der DOM danach neu aufgebaut. Diese bezeichnet man oft als Round-Trip Anwendungen.

Als weiteren Entwicklungsschritt wurde AJAX als Technologie populär. Hierbei werden bei Eintreten von Events im Hintergrund, ohne das *document* zu entladen, HTTP Requests an den Server geschickt und die Antwort wird in einer Callback Funktion manuell verarbeitet und der aktive DOM damit verändert.

Dieses dynamische nachladen von Daten eröffnet die Möglichkeit, auf volles Entladen des DOMs zu verzichten und möglichst wenige Navigationsvorgänge zu haben.

Bei Single Page Applications (SPA) verfolgt man die Idee, auf das komplette ent- und erneutes beladen des Document zu verzichten und im Sinne einer kompletten Webanwendung das gesamte Gerüst beim ersten Request in den Browser zu laden und danach nur noch mittels AJAX Daten und Teiltemplates nachzuladen.

Dadurch verlagert sich ein Großteil der Anwendungslogik in den Browser und wird nicht mehr in serverseitigen Programmiersprachen sondern in Javascript entwickelt. Klassische serverseitige Renderframeworks sind nicht mehr nötig, da der Server hauptsächlich zum Ausliefern von Daten genutzt wird.

Ist die Anwendung einmal in den Browser geladen, sind tendenziell weniger Serveraufrufe notwendig als bei klassischen Webseiten. Da die UI Logik im Browser bereits geladen ist und kaum HTML Fragmente ausgetauscht werden müssen, kann eine SPA für den Benutzer gefühlt flüssiger und angenehmer bedient werden. Man bekommt das Gefühl einer nativen Anwendung.

Durch Nutzung moderner Browserfunktionen wie der LocalStorage und IndexedDB können SPAs auch als mobile, offlinefähige Anwendung zum Einsatz kommen. Diese eignen sich auch für Servicearchitekturen sehr gut und erlauben es stark zu skalieren, wenn REST-API Aufrufe zustandslos genutzt werden können.

Da sehr viel Logik in den Browser verlagert wird, wächst der Umfang an Javascript Code und dessen Komplexität. Um dies zu beherrschen, wurden Javascript Frameworks entwickelt. Eines der beliebtesten Frameworks ist AngularJS bzw. Angular. Dieses wurde zur SPA Implementierung des *oekorr* Prototypen eingesetzt.

4.2 AngularJs

16 Logo angularjs.org

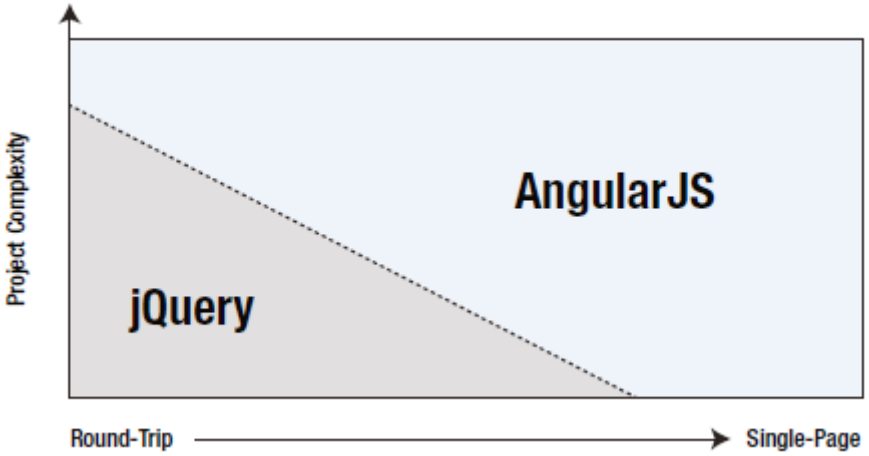
AngularJs ist ein clientseitiges Javascript Framework, welches rein im Browser ausgeführt wird. Ziel dieses Frameworks ist es, Tools und Möglichkeiten, die es bisher nur in serverseitigen Frameworks gab in den Browser zu bekommen. Weiters soll es damit einfacher werden, Weboberflächen und Anwendungen zu entwickeln und deren Wartbarkeit erhöhen.

HTML wurde zur Beschreibung von statischen Dokumenten entwickelt und ermöglicht es nur mühsam, dynamisch Änderungen der Anzeige vorzunehmen. Um Ansichten (Views) dynamisch zu ändern, muss mit Javascript der DOM manipuliert werden. Gewisse Änderungen kann man auch durch Definitionen in Cascading Style Sheets (CSS) erreichen.

AngularJS bietet nun die Möglichkeit, HTML durch eigene Tags und Attribute zu erweitern und damit Datenbindungen zu ermöglichen. Durch die Vermischung von Standard und Erweitertem HTML bleibt das Dokument weiterhin einfach zu lesen und kann rasch entwickelt werden.

Andere Bibliotheken wie jQuery erleichtern die Manipulation des DOM, bieten aber keine Möglichkeit ein Datenmodell und die graphische Oberfläche aneinander zu binden und Änderungen in beide Richtungen zu übernehmen. Bei jQuery werden immer mittels Selektoren DOM Elemente gesucht und danach verändert oder es werden durch Eventhandler Werte aus DOM Elementen ausgelesen und weiterverarbeitet. Dabei ist es aber sehr schwierig, DOM Elemente und Datenstrukturen synchron zu halten.

AngularJS ermöglicht es, komplette Anwendungen in Javascript nach dem Model-View-Controller Prinzip zu entwickeln.



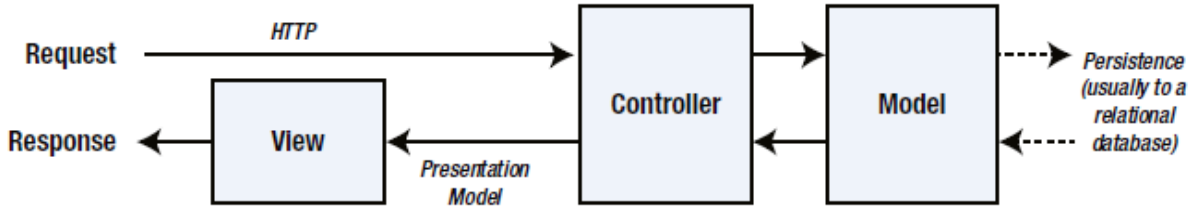
17 SPA mit AngularJS (18)

MVC ist ein altbekanntes Designpattern zur Handhabung von komplexen GUIs und wurde in den 1970er Jahren bei Xerox PARC entwickelt und in deren Smalltalk Projekt eingesetzt. Hierbei kommt es auf die Trennung von Verantwortlichkeiten an (*Separation of Concerns*). Damit meint man die Entkopplung von Daten- und Anzeigelogik. Für clientseitige Webanwendungen bedeutet dies, dass man die Datenmodelle, deren Verarbeitung und die erzeugten DOM Elemente (das erzeugte HTML) zur Darstellung voneinander trennt.

Durch MVC können clientseitige Anwendungen einfacher entwickelt, gewartet und getestet werden.

Der Einsatz von MVC in AngularJS unterscheidet sich vom Einsatz in klassischen Round-Trip Anwendungen. Folgende Grafik beschreibt eine serverseitige MVC Webanwendung, wie dies unter vielen anderen bei Microsofts ASP.Net MVC Framework der Fall ist. Ein HTTP Request wird vom Controller behandelt und dieser erzeugt das Model und bereitet Daten auf. Diese könnten in einer relationalen Datenbank gespeichert sein. Danach gibt der Controller die geladenen Daten an die View

weiter. Diese nutzt diese Daten in der grafischen Oberfläche, rendert gemeinsam mit dem Controller eine HTML Response und sendet diesen Text an den Browser des Clients.



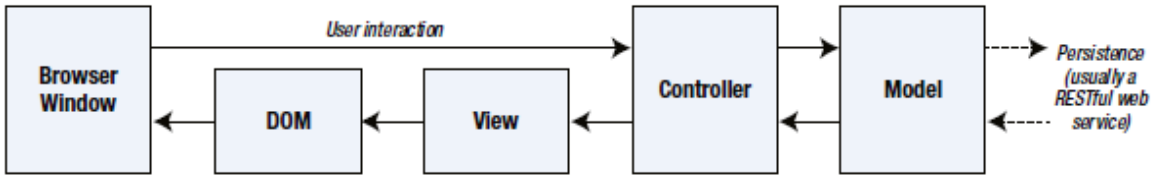
18 serverseitiges MVC (19)

Bei Single Page Applications, also rein clientseitigen (browserseitigen) Anwendungen wird das MVC Muster im Browser implementiert. Der Controller und die View arbeiten mit den Daten des Models. Das Model hängt meistens an einem RESTful Webservice und holt bzw. sendet Daten an dieses. Der Controller empfängt Benutzerinteraktionen (Events) und verarbeitet diese. Danach wird über die View der DOM im Browser verändert. Dabei werden neue HTML Elemente gezeichnet bzw. verändert und entfernt.

Im Unterschied zum klassischen MVC einer round-trip Anwendung lebt diese Anwendung im Browser und Benutzerinteraktionen führen immer wieder zu einem MVC Ablauf.

In AngularJS wird MVC auch sehr oft als MVW (Model-View-Whatever) bezeichnet. Mit Whatever meint man die Flexibilität von AngularJS. Dieses lässt sich als MVC aber auch als MVVM (Model-View-Viewmodel) einsetzen. Dies wird durch die Zweiwegedatenbindung ermöglicht. Jede Änderung der View kann im ViewModel manipuliert und dann sowohl im Model als auch in der View aktualisiert werden. MVVM ist sehr oft bei Mobilapplikationen oder modernen Desktop Frameworks wie WPF im Einsatz.

Um die Funktionsweise dieses Musters in einer SPA zu verstehen, werden nachfolgend die Zuständigkeiten der Bereiche erklärt. Das Model, welches am REST Service hängt ist als ein Domain Modell zu verstehen, welches Business Daten abbildet. Dazu gehören auch Methoden um die Daten zu erzeugen und zu manipulieren bzw. zu sortieren etc. Der Controller steuert den Flow, um das Business Modell und die View zu verbinden und zu kontrollieren. Durch den Controller werden die Daten an die Scopes gebunden. Die View sind hier bereits Attribute auf HTML Elementen, welche durch die AngularJs View an den DOM gebunden werden. Dabei werden Werte in DOM Elemente bzw. Controls geschrieben und von dort bei Änderung wieder ausgelesen. Dies passiert im Digest Cycle oder moderner durch OnPropertyChanged Handler. View und DOM sind lose gekoppelt.



19 MVC in AngularJS (19)

4.3 Deferred object, Futures und Promises

Moderne Webanwendungen sind interaktiv zu bedienen und bieten umfangreiche Daten und Informationen an. Diese Daten werden laufend über Web-Requests nachgeladen und präsentiert. Hierbei handelt es sich meist um AJAX Anfragen, welche dynamisch aus dem DOM heraus gestartet werden und deren Anfrageergebnis im Browser verarbeitet und damit der DOM bzw. das Document verändert werden.

Grundsätzlich sind Requests für die weitere Programmausführung immer blockierend, da auf eine Antwort gewartet werden muss. Dieses Warten kann synchron und asynchron stattfinden.

Bei synchronen Anfragen unterbricht die Javascriptengine die weitere Ausführung des Haupt-Threads und wartet auf das Ergebnis. Ist dieses eingetroffen oder ein Fehler aufgetreten, arbeiten die Eventhandler das Anfrageresultat weiter ab. In der Zwischenzeit treten keine weiteren Ereignisse ein und der Browser wirkt eingefroren. Bei klassischen Webseiten werden Requests synchron abgearbeitet. Der Benutzer klickt auf einen Link und wartet auf die Antwort des Webservers. Bei dynamischen Webanwendungen möchte man die Anwendungsmöglichkeiten und die Erfahrung für den Benutzer verbessern und blockierende Ereignisse möglichst vermeiden, weshalb Daten meist asynchron nachgeladen werden.

Bei asynchronen Anfragen startet der Browser eine Anfrage an einen Endpunkt, blockiert danach die Ausführung aber nicht bis eine Antwort eingetroffen ist, sondern registriert lediglich Eventhandler, die später bei Eintreten eines Ereignisses aufgerufen werden.

Diese Eventhandler bezeichnet man auch als Callbacks. Je größer und umfangreicher eine Anwendung wird, desto schwieriger wird es die Aufrufe und deren Callbacks zu strukturieren und die Anwendung wartbar zu halten.

Bei klassischen Vanilla-Javascript oder jQuery Calls kann es leicht passieren, dass man zum Laden von voneinander abhängigen Daten Funktionen ineinander verschachtelt. Dabei verwendet man sehr oft anonyme Funktionen. Diese sind stark miteinander gekoppelt und schwer wartbar.

An folgendem Beispiel sieht man die starke Koppelung und eine starke Verschachtelung, die man noch weiterführen könnte. Diese bezeichnet man auch als Callback-Hell (Callback-Hölle):

```
1.  getData = function(param, callback){
2.    $.get('http://example.com/get/'+param,
3.      function(responseText){
4.        callback(responseText);
5.      });
6.  }
7.
8.  getData(0, function(a){
9.    getData(a, function(b){
10.     getData(b, function(c){
11.       getData(c, function(d){
12.         getData(d, function(e){
13.          // ...
14.         });
15.       });
16.     });
17.   });
18. });
```

(20)

Es wird eine Funktion getData definiert, welche eine Zahl (Typ number) als Parameterwert und eine Funktionsreferenz als Callback übergeben bekommt. Die getData Funktion erzeugt dann über das jQuery-Framework einen GET http Request und registriert den Callback, welcher bei Eintreten eines Ereignisses aufgerufen wird. Diese getData wird dann mit dem Parameterwert „0“ und einem Callback aufgerufen und dieser Callback ruft dann mehrmals als Callback getData mit dem Antworttext (responseText) auf.

Dies ist ein stark vereinfachtes, aber durchaus praktisch relevantes Beispiel und zeigt deutlich, dass es sehr schwierig werden kann diese Konstrukte zu durchschauen und zu warten.

Um diese Handhabung in Javascript zu vereinfachen, wurden mit EMCAScript 2015 Promises in Javascript eingeführt und diese werden mittlerweile von allen gängigen Browsern nativ unterstützt. Als Fallback bietet die jQuery Bibliothek seit Längerem ein deferred object, das sich analog zu Promises verhält.

Ursprünglich, in anderen Programmiersprachen wie Scala werden Promises auch Features genannt.

> *The construct (future X) immediately returns a future for the value of the expression X and concurrently begins evaluating X. When the evaluation of X yields a value, that value replaces the future.*
>
> *(21)*

Wie der Name es schon vermuten lässt, verspricht dieses Konstrukt in der Zukunft bei Eintritt eines Ereignisses oder Fehlers zu reagieren. Bei der Erzeugung wird ein Promise Objekt instanziert, welches später durch den tatsächlichen Wert ersetzt wird. Dies passiert im ereignisgetriebenen Javascript über zuvor registrierte Callbacks.

Da Javascript grundsätzlich nur in einem Thread ausgeführt wird, setzt man auf ein Event-Loop-System. Hierbei werden Nachrichten von Hilfsthreads wie Zeitgeber-, Netzwerk- und I/O-Thread in eine Warteschlange gestellt und immer, wenn der Hauptthread die Ausführung unterbricht, wertet das Event-Loop-System Nachrichten aus und stellt möglicherweise einen hinterlegten Callback auf den Funktionsstack, damit dieser vom Hauptthread ausgeführt wird.

Promises in Javascript sind asynchron und lassen sich verketten. Es wird garantiert, dass ein Callback nicht unmittelbar bei der Zuweisung ausgeführt wird, sondern immer zuvor in der Nachrichtenwarteschlange landet. Jedes Promise darf nur ein einziges Mal reagieren und einen Callback auslösen.

Folgendes Beispiel zeigt die vorangehende Callback-Hölle nun als Promises implementiert:

```
1.  getData = function(param, callback){
2.    return new Promise(function(resolve, reject) {
3.      $.get('http://example.com/get/'+param,
4.      function(responseText){
5.        resolve(responseText);
6.      }).fail(reject);
7.    });
8.  }
9.
10. getData(0).then(getData)
11.   .then(getData)
12.   .then(getData)
13.   .then(getData);
```

Der zur Implementierung notwendige Code ist deutlich vereinfacht und leichter verständlich.

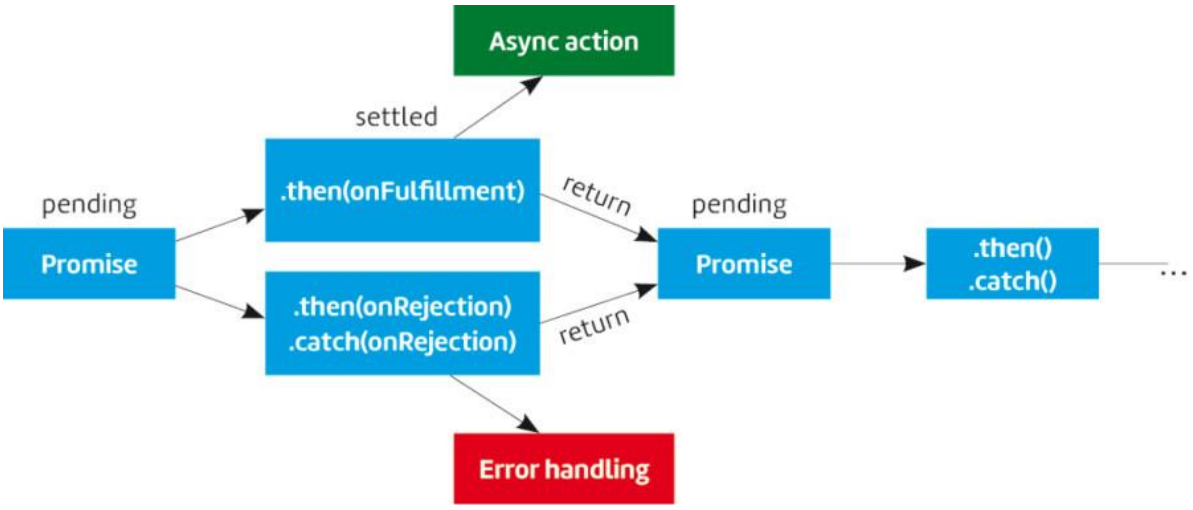
Im Erfolgsfall wird die Callback-Funktion *resolve* und im Fehlerfall die Callback-Funktion *reject* aufgerufen.

Man sieht auch, wie Promises miteinander verkettet werden können.

Ein Promise kann folgende Zustände annehmen:

- Pending: Promise wurde erzeugt, aber noch nicht abgearbeitet.
- Fullfilled: Promise wurde fertig erfüllt und Callback-Nachricht auf die Warteschlange gelegt.
- Rejected: Promise wurde wegen eines Fehlers (Exception) nicht erfüllt, aber fertig ausgeführt.
- Settled: Promise wird gerade abgearbeitet.

Folgende Grafik illustriert dies:



20 Promise Flow (22)

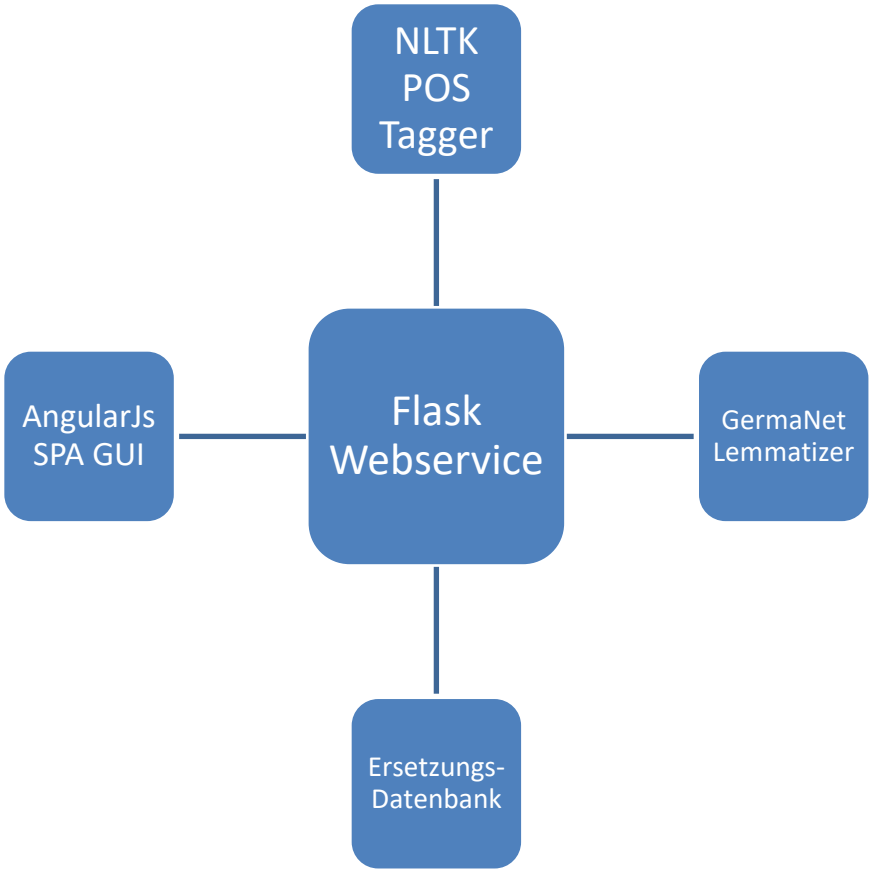
jQuery implementiert Deferred-Objects. Diese sind abwärtskompatibel zu älteren Browsern. Es implementiert eine eigene Warteschlange, die sich ähnlich zu Promises verhält und entweder im Erfolgs- oder Fehlerfall reagiert. Deferred Objekte werden nicht nativ vom Browser verarbeitet, sondern im DOM als .data() an einen Knoten gehängt. Sollte dieser Knoten während der Ausführung/Wartezeit eines Deferred Objects aus dem DOM entfernt werden, können die Callbacks nicht mehr ausgeführt werden und gehen verloren.

5 Oekorr

In den vorherigen Kapiteln wurden Technologien eingeführt und beschrieben, welche bei der Implementierung des *oekorr* Webservices und der Webanwendung zum Einsatz kamen.

In diesem Kapitel wird die Funktionsweise beschrieben und vorgeführt.

5.1 Übersicht der Teile



21 Vereinfachte Komponenten

5.2 Kernanwendung und Webservice

Das Webservice wird von einem in Python entwickelten Mikroframework betrieben, welches FLASK (10) heißt. Dieses bietet Basisfunktionen zum Abhandeln von Requests und ermöglicht eine sehr flexible Implementierung von Webanwendungen und Services. In der Oekorr Anwendung wird sowohl die SPA als auch das Webservice in der selben Flask Instanz betrieben.

Hierzu liefert FLASK die statische HTML Seiten aus, welche die Single Page Application beinhält. Nachdem die Index Page im Browser angekommen ist, initialisiert AngularJS die SPA und lädt alle weiteren Templates und statischen Dateien durch FLASK.

Zur Sprachverarbeitung kommt das Natural Language Toolkit (NLTK) zum Einsatz, welches vom Webservice genutzt wird. Dieses wird in Python geschrieben und soll vom Webservice genutzt werden. Da dies direkt in Python einfach eingebunden werden kann, musste ein Python Webframework genutzt werden und die Wahl des serverseitigen Frameworks fiel auf FLASK.

FLASK kann sehr flexibel erweitert werden und es gibt sowohl gute Zugriffsbibliotheken zu MySql als auch MongoDB Datenbanken. Beide Datenbanksysteme kommen bei Oekorr zum Einsatz.

Das NLTK wird genutzt, um die eingegebenen Texte zu tokenisieren. Als Tokenizer wird der Punkt Tokenizer (*Punkt sentence tokenizer*) verwendet und mit einem Pickel für die deutsche Sprache initialisiert. Nach dem Tokenisieren des Textes zu Sätzen werden die Sätze weiter in einzelne Wörter zerlegt. Dabei werden die Stopwords entfernt. Dies sind einfache Wörter, deren weitere Verarbeitung keinen Mehrwert bringen. Dafür wird ein Korpus von NLTK für Deutsch eingesetzt.

Danach werden die Tokens mit Tags versehen. Dazu wird der *ClassifierBasedGermanTagger* (23) als Erweiterung des NLTK eingesetzt. Hierbei handelt es sich um einen überwachten Klassifizierer (*Supervised Classifier*). Dazu muss der Klassifizierer trainiert werden und dabei ein Modell aufbauen. Einer der umfangreichsten Korpora mit deutscher Sprache ist der Tiger Korpus der Universität Stuttgart.

Zum Einlesen des Korpus wird der *NegraCorpusReader* (24) des NLTK genutzt. Der Tiger Korpus wird neben dem CONLL09 Format auch im NEGRA Format bereitgestellt, was diesem sehr ähnelt. Jedes Wort und dessen POS-Tag und Lemma werden in einer Zeile dargestellt.

```
%% word     lemma       tag     parent
#BOS 1
The         the         DET     500
house       house       N       500
is          be          V       501
red         red         ADJ     501
.           --          .       502
#500        --          NP      502
#501        --          VP      502
#502        --          S       0
#EOS 1
```

22 NEGRA Format Beispiel (24)

Hier wird nun der *ClassifierBasedGermanTagger* mit dem TIGER Korpus initialisiert. Mit 90% des TIGER Korpus konnte durch die Validierung mit den verbleibenden 10% des Korpus eine Genauigkeit von über 92% erreicht werden.

Das Trainieren des Klassifizierers nimmt je nach Rechner viel Zeit, zumindest einige Minuten in Anspruch. Dies bei jeder Verwendung zu machen dauert zu lange und bremst die Anwendung sehr stark. Damit der Tagger schneller initialisiert werden kann, ermöglicht es das NLTK, dieses initialisierte Modell als Speicherabbild zu serialisieren. Diese Datei nennt man ein Pickel. Damit kann das Modell schnell wieder geladen und verwendet werden.

Nachfolgend nun als Beispiel die Verwendung des NLTK in *oekorr* zum Trainieren und Evaluieren des Taggers:

```
def trainTagger(self):
    """
    trains a ClassifierBasedGermanTagger with 90% of TIGER corpus getting
an accuracy of about >92%
    not used in production, may except
    """
    path = nltk.data.find('corpora/tiger/')
    reader = NegraCorpusReader(path, 'tiger_release_aug07.export')

    # tagged_words = reader.tagged_words()
    tagged_sentences = reader.tagged_sents()
    logger.debug("corpus loaded.")

    tagger = ClassifierBasedGermanTagger(train=tagged_sentences[:45000])

    f = open(self.__filePathToPickle, 'wb')
    pickle.dump(tagger, f)
    f.close()

    logger.debug("trained and pickle saved.")
```

```
def evaluateTagger(self):
    """
    evaluates a ClassifierBasedGermanTagger with 10% of TIGER corpus
getting an accuracy of about >92%
    not used in production, may except
    """
    path = nltk.data.find('corpora/tiger/')
    reader = NegraCorpusReader(path, 'tiger_release_aug07.export')

    # tagged_words = reader.tagged_words()
    tagged_sentences = reader.tagged_sents()
    logger.debug("corpus loaded.")

    f = open(self.__filePathToPickle, 'rb')

    tagger = pickle.load(f)
    logger.debug("pickle loaded. starting to evaluate...")

    accuracy = tagger.evaluate(tagged_sentences[45000:])

    logger.debug("Accuracy: " + "{:.4f}".format(accuracy))

    return accuracy
```

Hier werden 45000 Sätze zur Evaluierung der Genauigkeit des Taggers genutzt. Dies entspricht ca. 10 % des gesamten Korpus.

Der German Tagger versieht nun jedes Token mit einem Tag. Der TIGER Korpus ist sehr umfangreich getaggt und um die weitere Verarbeitung zu vereinfachen, wurden in *oekorr* Tags zusammengefasst. Folgende Gruppierung wird verwendet:

```
nounsTags = ['NN', 'NE']

adjectivesTags = ['ADJA', 'ADJD']

verbsTags = ['VVFIN'
    , 'VAFIN'
    , 'VMFIN'
    , 'VVINF'
    , 'VAINF'
    , 'VMINF'
    , 'VVIMP'
    , 'VAIMP'
    , 'VVPP'
    , 'VAPP'
    , 'VMPP'
    , 'VVIZU']
```

Danach wird GermaNet verwendet, um für jedes Token das Lemmata, also die Stammform zu finden. Erst durch das Auffinden der Stammform kann man später eine Wortersetzung in einem Wörterbuch finden.

GermaNet wurde mit *pygermanet* (25) aus den bereitgestellten XML Dateien in eine MongoDB Datenbank geschrieben und Oekorr durchsucht nun dort alle Synsets, um Lemmata zu finden.

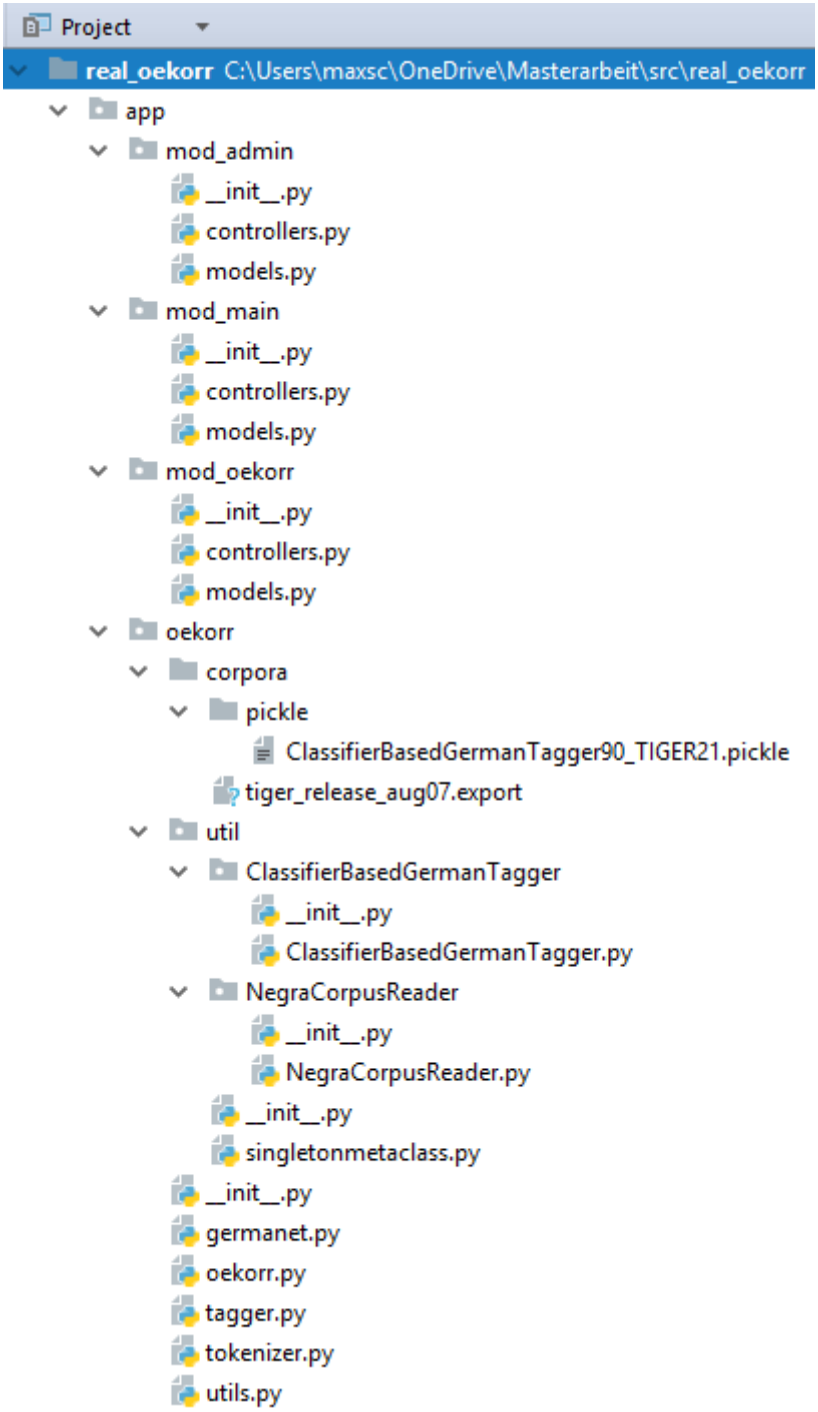
Das Wörterbuch zur Wortersetzung wird in einer MySql Datenbank gespeichert. In dieser werden die Wörter nach Substantiven, Verben und Adjektiven kategorisiert. Dieses wurde initial mit einer Sammlung an Tautonismen befüllt und kann über eine GUI erweitert und verändert werden.

Für jedes Lemmata wird dann gemeinsam mit der POS Information nach einer Wortersetzung in der Datenbank gesucht. Wenn eine Ersetzung gefunden wurde, dann wird diese vorgeschlagen. Zusätzlich zum Vorschlag werden auch die POS Informationen präsentiert.

Nun kann man diese Informationen nutzen, um das Wort von deutschländischem Deutsch in österreichisches Deutsch zu übersetzen.

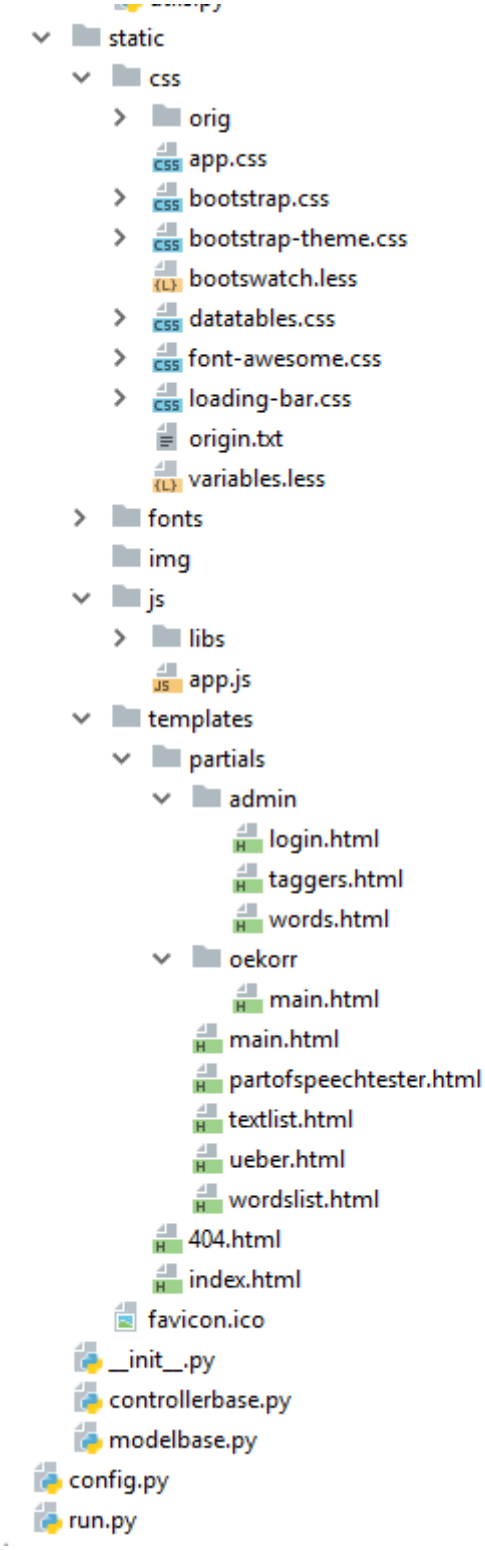
5.3 Flask Struktur von real_oekorr

In Flask gibt es drei Module, welche je aus Controller und Modells bestehen. Die Sprachverarbeitung wurde in Utils ausgelagert. Beim Start der Anwendung werden GermaNet und der POS Tagger geladen und sind durch das Singleton Pattern aus allen Controllern zugreifbar.



23 oekorr Projekt in pyCharm IDE Teil 1

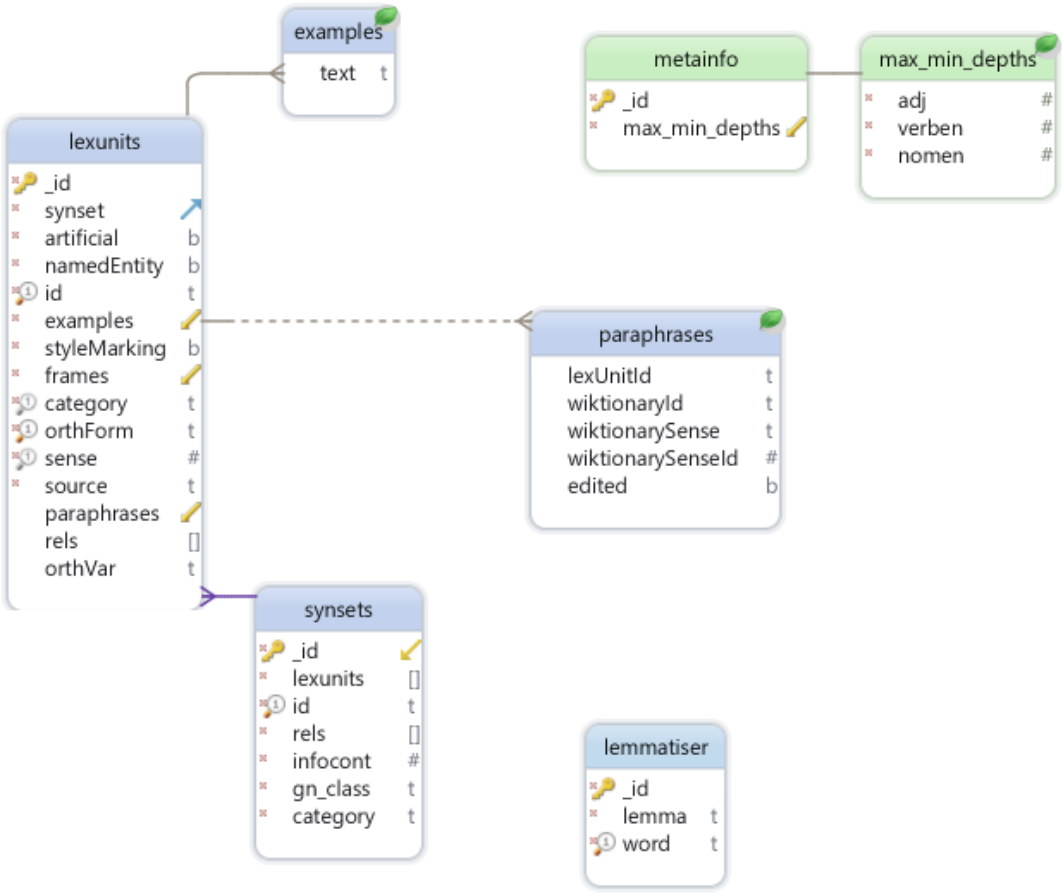
Die statischen Dateien und Templates befinden sich in separaten Ordnern.



24 oekorr Projekt in pyCharm IDE Teil 2

5.4 Datenstrukturen

Zum Finden der Lemmata wird GermaNet in einer MongoDB Datenbank verwendet. Dabei wird über *words* nach *lemma* im *lemmatiser* gesucht. Folgende Datenstruktur bildet dies ab.



25 GermaNet in MongoDB

Die Wortersetzungen werden in einer MySql Datenbank gespeichert und über eine Stored Procedure in dieser gesucht. Folgende simple Datenstruktur wird dafür genutzt.

26 MySQL Datenbankstruktur der Ersetzungen

5.5 Webservice API

Das REST Service umfasst einige Methoden, wovon die wichtigsten nachfolgend gelistet sind.

5.5.1 Wortersetzungen

Mit diesem Aufruf können alle aktuellen Wortersetzungen ausgelesen werden. Die GET Methode ist öffentlich, ohne Login verfügbar. PUT, POST, DELETE erfordern ein gültiges JWT Token im Authorization: Bearer <token> HTTP Header.

Als Beispiel der GET Request mit Response:

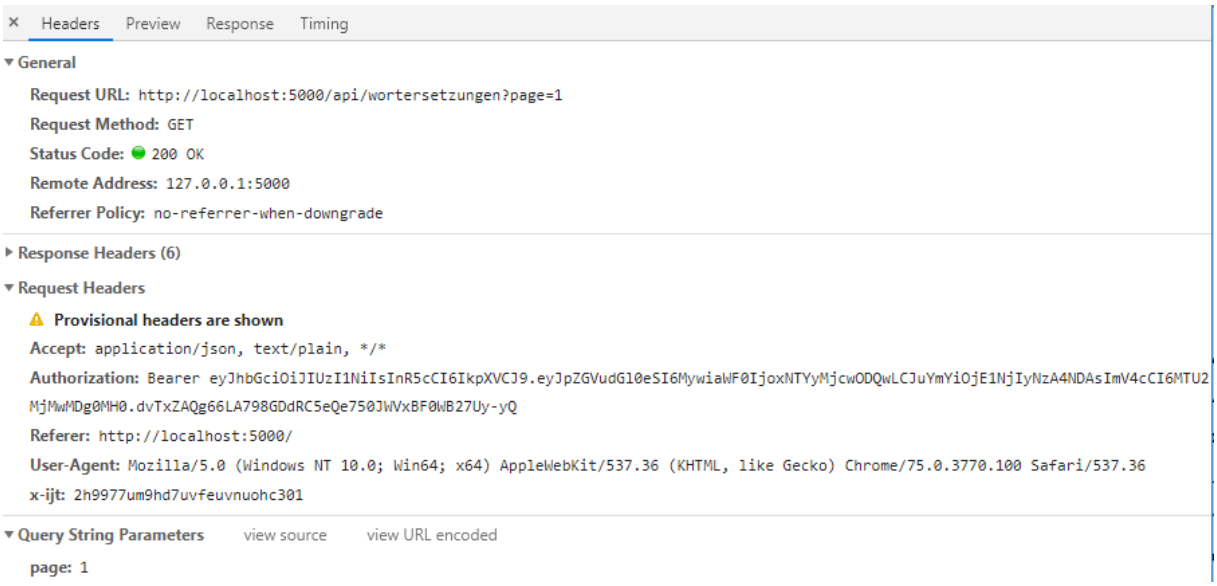
GET:

http://localhost:5000/api/wortersetzungen?page=1

Response:


```
num_results: 1382
objects: [{at: "abrebeln", at_genus: "", date_created: null, date_modified: "2019-04-24T17:31:00",…},…]
  0: {at: "abrebeln", at_genus: "", date_created: null, date_modified: "2019-04-24T17:31:00",…}
  1: {at: "abschreiben", at_genus: null, date_created: null, date_modified: null, de: "abbimsen",…}
  2: {at: "Erstklässler", at_genus: "m", date_created: null, date_modified: null, de: "ABC-Schütze",…}
  3: {at: "Abendessen", at_genus: "n", date_created: null, date_modified: null, de: "Abendbrot",…}
  4: {at: "Mistkübel", at_genus: "m", date_created: null, date_modified: null, de: "Abfalleimer",…}
  5: {at: "Abfertigung", at_genus: "f", date_created: null, date_modified: null, de: "Abfindung",…}
  6: {at: "abgenutzt", at_genus: null, date_created: null, date_modified: null, de: "abgeledert",…}
  7: {at: "abschauen", at_genus: null, date_created: null, date_modified: null, de: "abgucken",…}
  8: {at: "abschauen", at_genus: null, date_created: null, date_modified: null, de: "abkucken",…}
  9: {at: "Matura", at_genus: "f", date_created: null, date_modified: null, de: "Abitur", de_genus: "n",…}
page: 1
total_pages: 139
```


27 Debug Tools in Google Chrome mit einem JSON Result



28 Debug Tools in Google Chrome mit dem HTTP Request Header

5.5.2 Texte

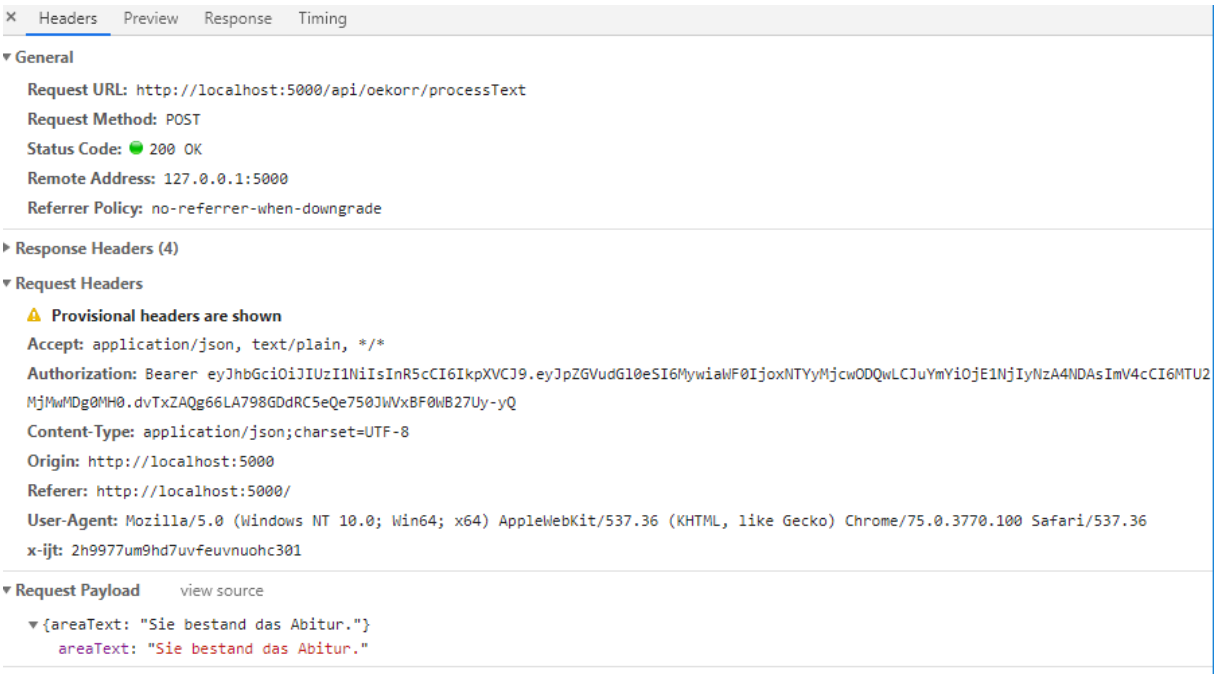
Eine CRUD Methode die das Abspeichern von Texten ermöglicht.

5.5.3 Auth

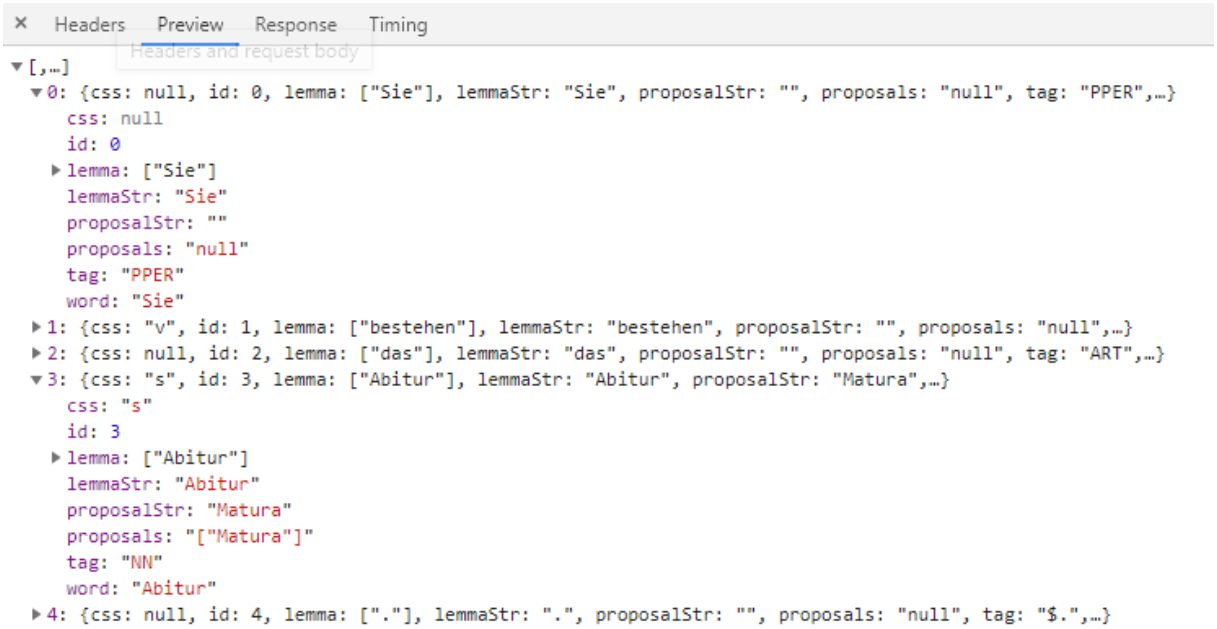
Damit werden als POST Benutzername und Kennwort übermittelt und bei Erfolg wird ein JWT Token retourniert.

5.5.4 ProcessText

Diesem POST Request wird der zu analysierende Text übergeben und das Service antwortet mit Vorschlägen. Folgende Screenshots zeigen dies beispielhaft.



29 Request an das Ersetzungsservice

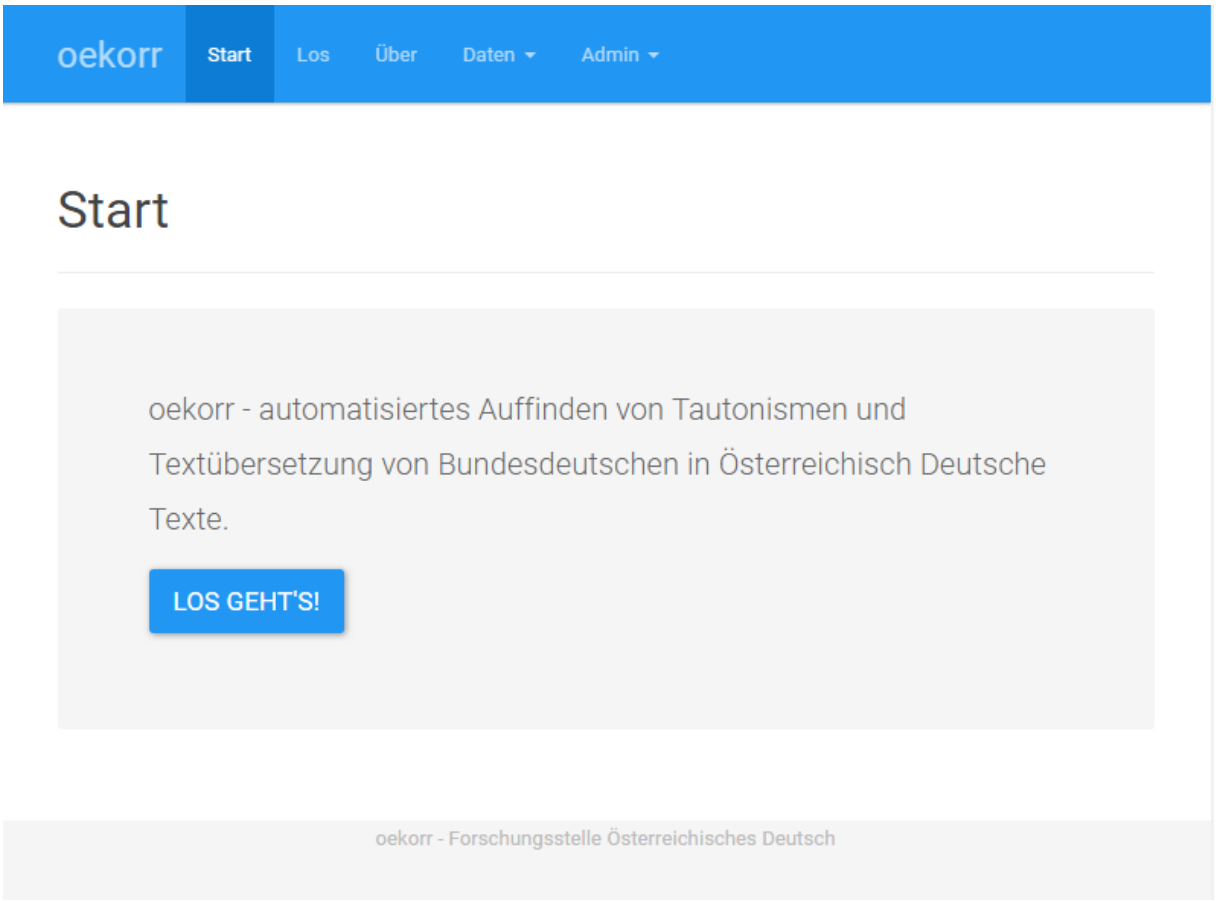


30 Response des Ersetzungsservice

5.6 Web GUI

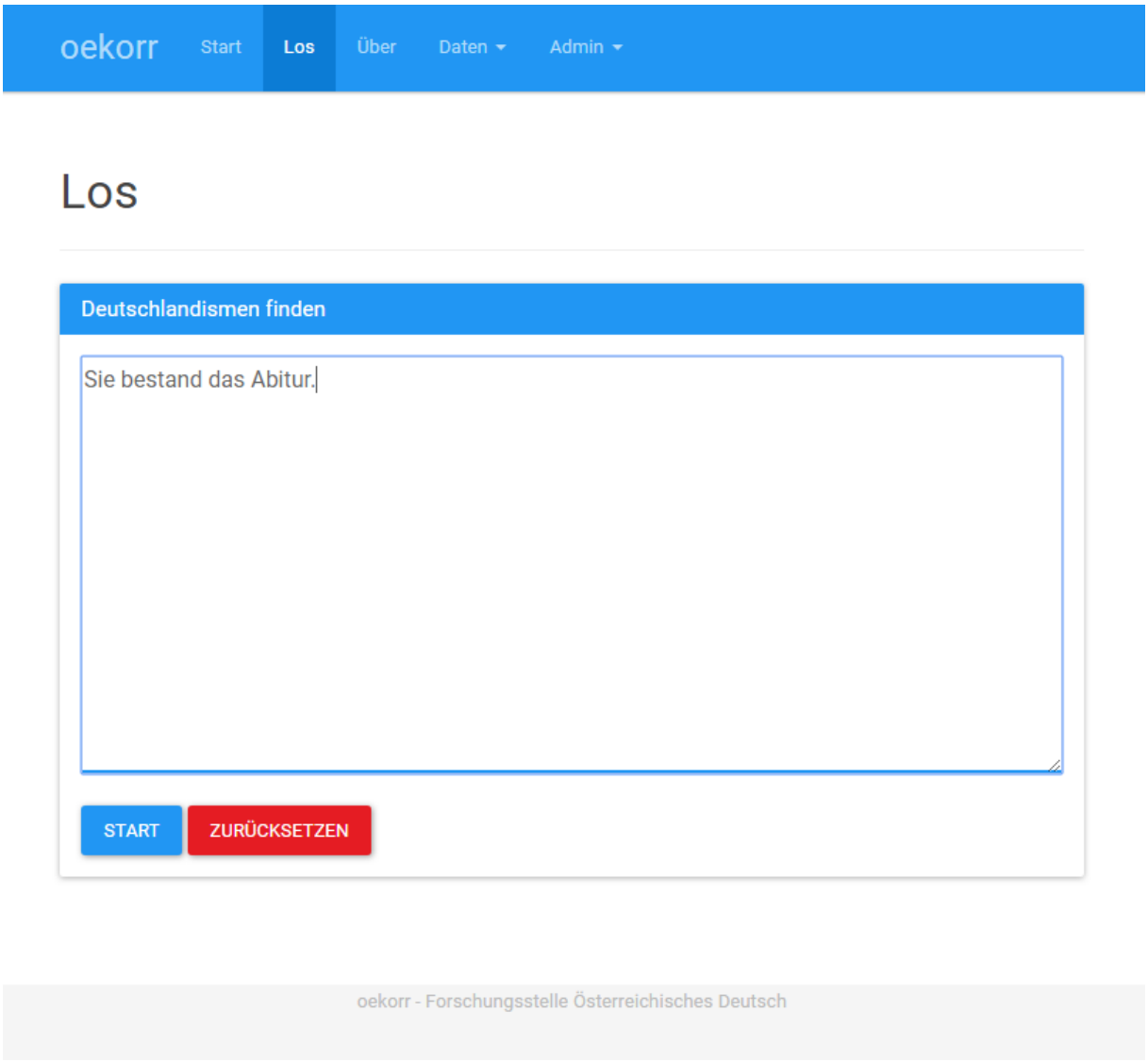
Die Webanwendung wurde als Single Page Application mit AngularJS implementiert. Als Layoutframework kommt Bootstrap (26) zum Einsatz. Der Look und Feel der Webanwendung kommt dem Material Design nahe. Auf kleinen Bildschirmen wird die Hauptmenünavigation durch ein Burgermenü ersetzt, wie es bei responsiven Webseiten üblich ist. Das statische HTML File wird auch durch Flask geliefert und danach von AngularJs gesteuert. Diese Anwendung besteht aus sechs Controllern und fünf Services. Sämtliche Daten werden über Webservice Aufrufe geliefert. Dafür wird Angulars' $http Implementierung genutzt. Zur Authentifizierung kommen JSON Web Token zum Einsatz, welche in den $http Request intercepted werden. Der JWT Token String wird in der Local Storage des Browsers abgelegt und ist somit bis zum Ablaufen der Gültigkeit verfügbar. Es werden möglichst wenig Daten im Browser gehalten. Deshalb wurde für alle Datengrids ein Paging eingebaut und es wird jede Page separat vom REST Service geladen.

Nachfolgend befinden sich Screenshots der einzelnen Seiten, durch welche navigiert werden kann.



31 Oekorr Landingpage

In einer Textarea kann beliebig viel Text eingegeben werden. Dieser wird dann im NLP Webservice verarbeitet.

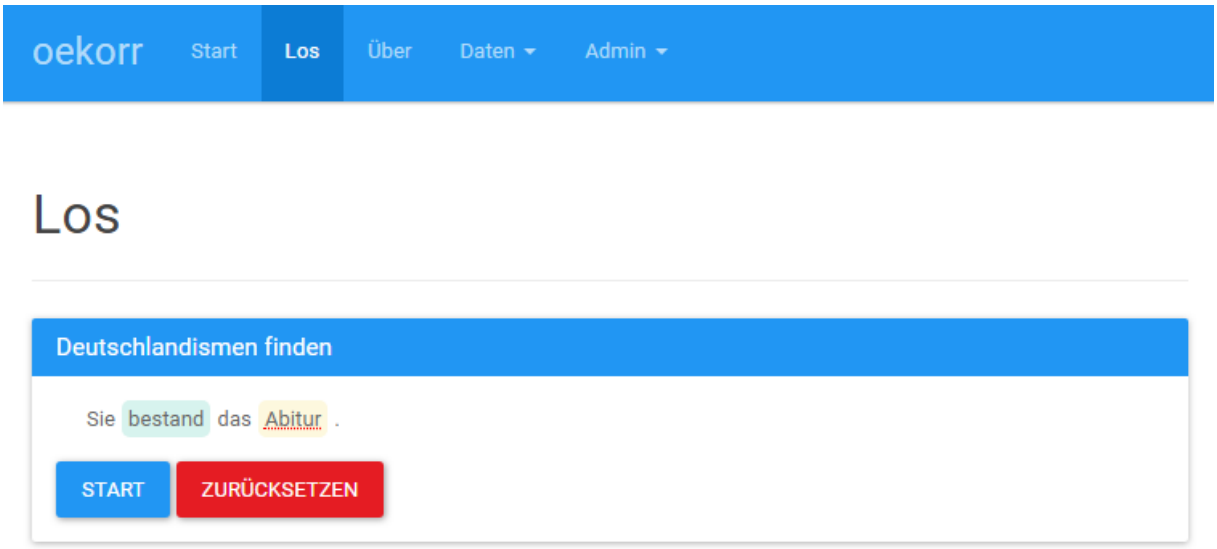


32 Textarea zur Eingabe

Das Ergebnis des NLP Webservice wird danach ausgegeben. Dabei werden die Wörter nach ihren POS Tag Kategorien farblich markiert.

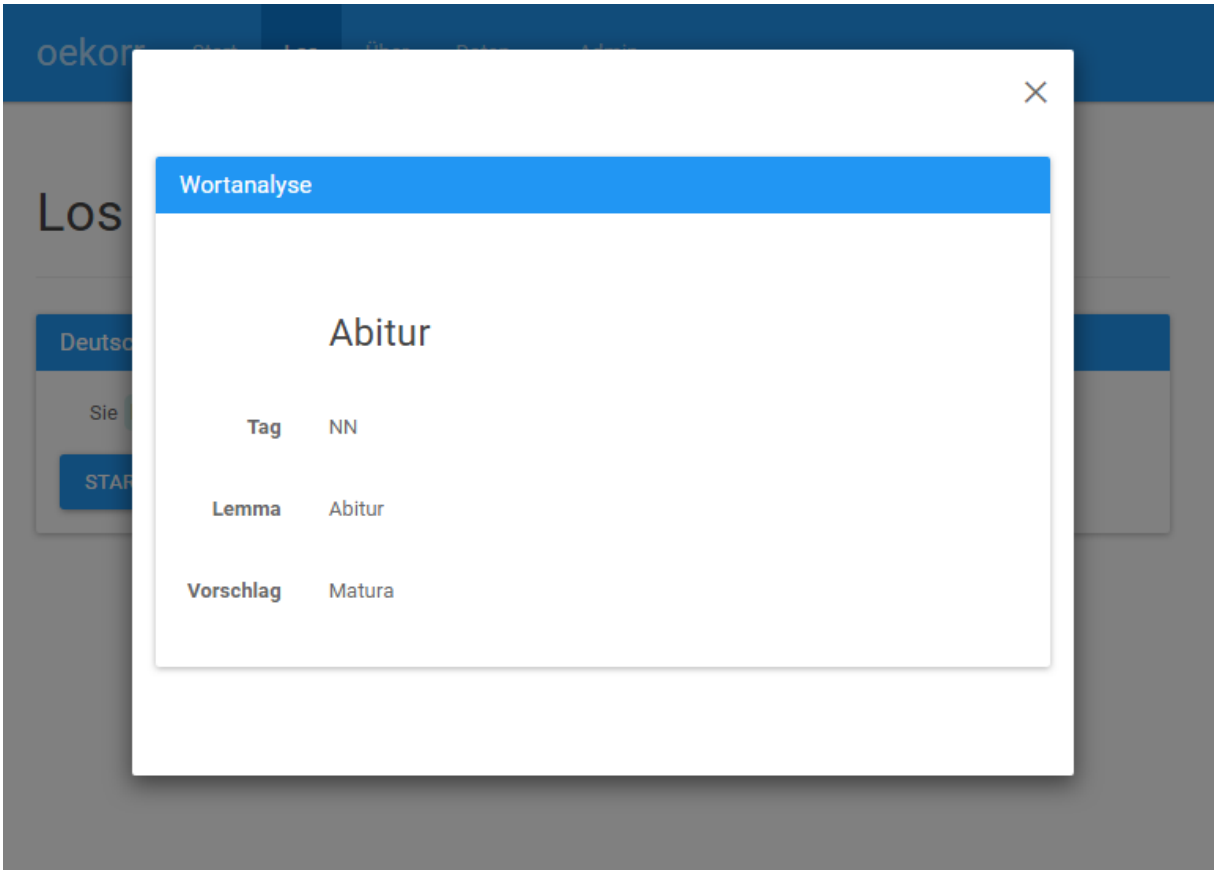
- Verben in grün
- Substantive in gelb
- Adjektive in rot

Wenn ein Ersetzungsvorschlag gefunden wurde, wird das Wort rot unterstrichen.



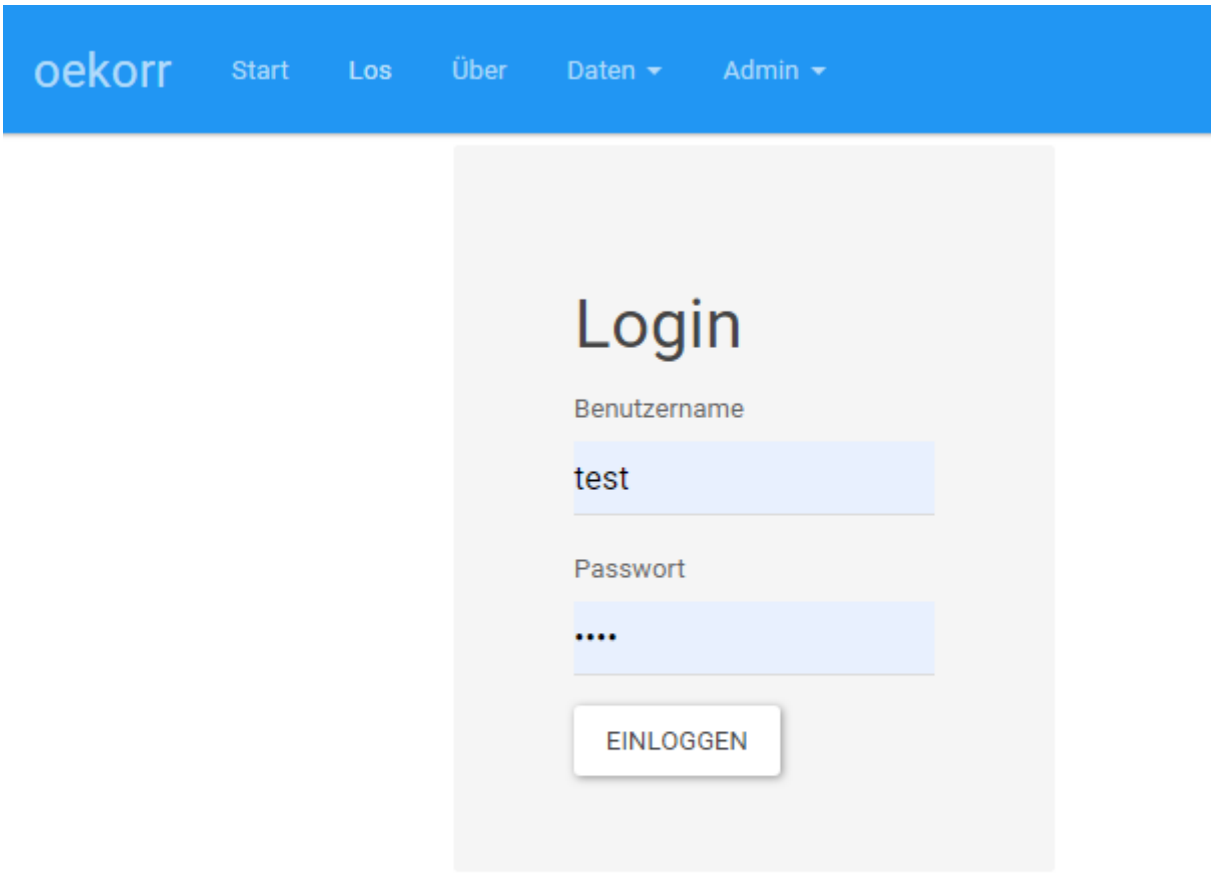
33 Ergebnis der Analyse

Mit einem Klick auf das Wort kommt man zu den Analysedetails. Hier wird das gefundene POS Tag, das Lemmata und der Korrekturvorschlag angezeigt.



34 Detailergebnis der Analyse

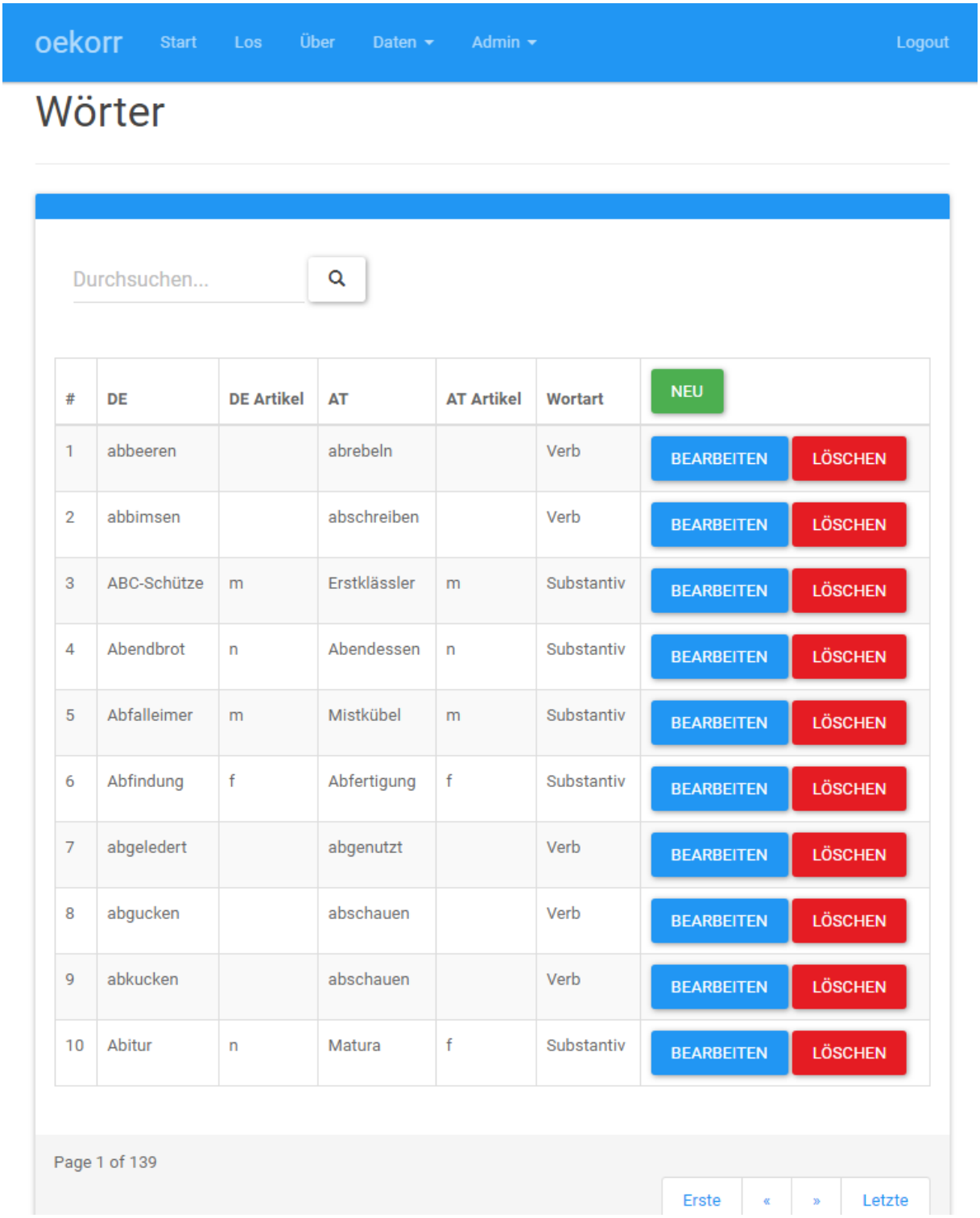
Um *oekorr* erweitern zu können und aus diesem Basis-Prototypen ein umfangreich einsetzbares Tool zu machen, kann man die Wortersetzungen in einem Administrationsbereich warten. Diese Login Seite validiert über einen Service Aufruf die Zugangsdaten und bekommt vom Webservice einen JWT Token. Dieses wird dann für weitere Aufrufe verwendet.



35 Login zum Adminbereich

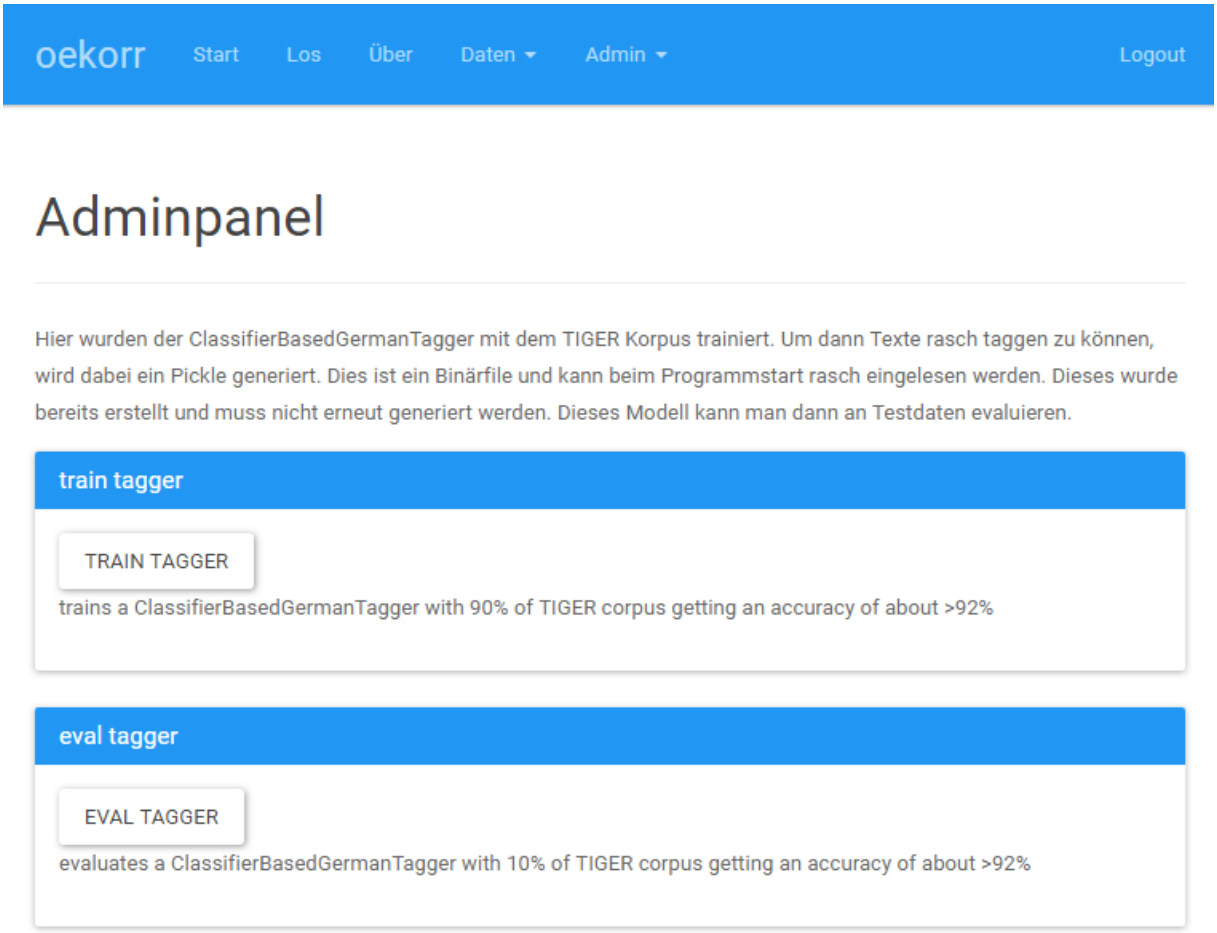
Hier können die Wortersetzungen gewartet werden. Es können Bestehende bearbeitet und gelöscht und Neue eingefügt werden. Dieses AngularJS Service verwendet eine REST API mit allen CRUD (Create, Read, Update, Delete) Methoden.

Die Datenbank kann auch durchsucht werden.



36 Verwaltung der Wortersetzungen

Wie eingehend beschrieben, muss der Klassifizierer zum Erkennen der der POS Tags trainiert und evaluiert werden.



37 Klassifizierer trainieren

6 Evaluierung

Die Herausforderung bei der Entwicklung dieses Prototyps war die Findung unterschiedlicher Lösungsansätze und die anschließende Bewertung dieser. Nachfolgend werden eingesetzte Tools diskutiert.

6.1 Frameworks

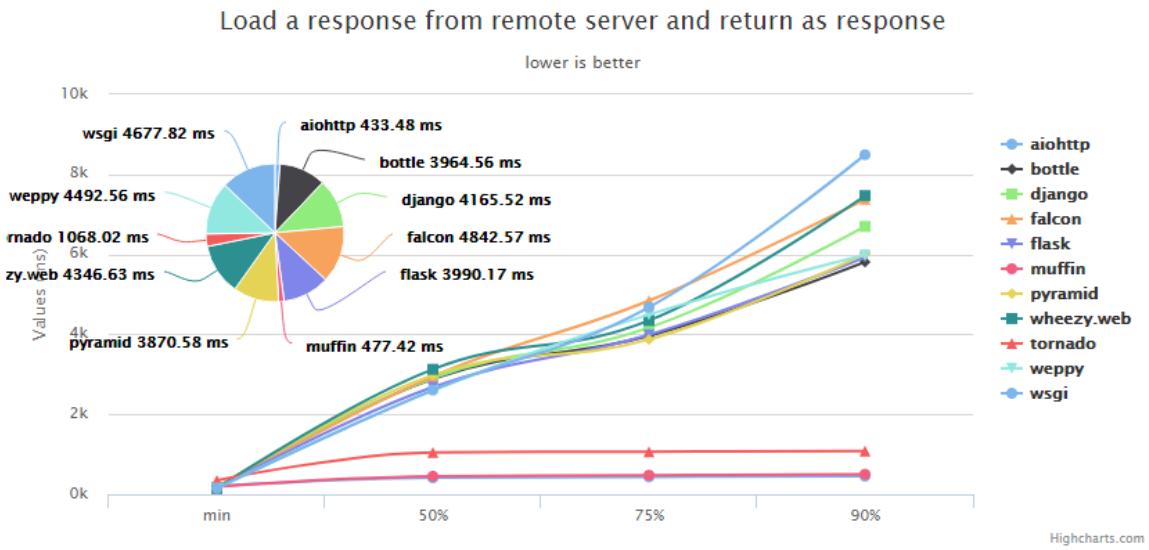
Mit Flask wurde ein leichtgewichtiges WebAPI Framework eingesetzt. Im Performance Vergleich zu anderen Python Webframeworks befindet sich Flask im Mittelfeld. Da für *oekorr* aber nur eine mittlere Anzahl an Anfragen pro Sekunde erwartet werden können, war Performance nicht das wichtigste Kriterium. Es wurde anderen Frameworks wie Django dem Vorzug gegeben, da es weniger Funktionalität in der Grundkonfiguration mitbringt und die Lernkurve geringer ist. Dadurch kann es besser beherrscht werden. Durch die Bereitstellung vieler Bibliotheken kann Flask aber einfach an Grundfunktionalität erweitert werden. Es stehen Treiber zu allen gängigen Datenbanken zur Verfügung.

Ein Benchmark Ergebnis der Performance zeigt Abbildung 38.

Flask ist eigentlich ein starker HTML Renderer. Die Jinja Template Rendering Engine wird mitausgeliefert. Damit ist man aber vor allem im Nutzen von klassischen Round-trip Webanwendungen stark. Wie vorhergehend in der Literatur erklärt wurde, verfolgt man heute den Einsatz von MVVM Frameworks im Browser. Deshalb wird die Rendering Engine nicht genutzt und Flask stellt lediglich die WebAPI bereit.

Um MVVM im Browser nutzen zu können, wurden clientseitige Frameworks evaluiert und AngularJS eingesetzt. Dieses erfüllt das Kriterium eines JSON Models im Browser, Zweiwegedatenbindung und die einfache Nutzung von REST WebAPIs mit Boardmitteln. Es ist keine weitere Javascript Bibliothek für *oekorr* notwendig, da AngularJS alle relevanten Funktionalitäten bietet. Angular wird von namhaften Unternehmen unterstützt und weiterentwickelt und ist ein zukunftssicheres Framework. Für oekorr wurde noch AngularJS eingesetzt. Dieses wurde mittlerweile in Angular umbenannt und die in oekorr verwendete Version sollte in Zukunft aktualisiert werden.

Webanwendungen können auch auf Tablets und Smartphones genutzt werden. Um dies zu ermöglichen wurde nach einem CSS Framework gesucht und Bootstrap eingesetzt. Mit dessen Grid Layout System ist die Webanwendungen bei unterschiedlichen Displaygrößen adaptiv und responsiv.



Name	50% (ms)	75% (ms)	Avg (ms)	Req/s	Timeouts
Aiohttp	412.08	433.48	406.53	483	615.20
Muffin	451.77	477.42	449.69	439.1	617.90
Tornado	1044.8	1068.02	1036.97	187.8	103.65
WSGI	2607.26	4677.82	3578.6	18.4	16.15
Falcon	2944.88	4842.57	3629.31	18.35	14.20
Wheezy.Web	3125.03	4346.63	3573.43	18.35	13.70
Bottle	2885.36	3964.56	3207.14	18.3	15.85
Weppy	2593.71	4492.56	3468.05	18.25	16.15
Flask	2679.56	3990.17	3344.27	18.15	13.25
Django	2908.53	4165.52	3477.36	18.1	14.30
Pyramid	2980.71	3870.58	3305.18	18	14.10

38 Python Frameworks Durchsatz an Anfragen (27)

6.2 Wortschatz

Die inhaltliche Basis dieses Korrekturtools liefert dessen Ersetzungswörterbuch. In Zusammenarbeit mit der Forschungsstelle Österreichisches Deutsch wurde von Dr. Rudolf Muhr ein Wortschatz aufbereitet, welcher initial in dieses Wörterbuch geschrieben wurde.

Hierfür wurden Grubers Piefke-Wörterbuch von Reinhard Gruber, Auszüge aus Deutsch-Rumänisch Wörterbuch (LAZ/SCHEUR) sowie einige eigene Sammlungen verwendet. Diese ergeben gemeinsam einen Wortschatz von ca. 3500 Wörtern. Dies stellt eine solide Basis da und kann im *oekorr* Tool jederzeit erweitert werden.

In Tests wurden Texte aus österreichischen Tageszeitungen verwendet. Vorwiegend aktuelle Artikel aus *Kleine Zeitung*, *Die Presse* und *Der Standard*.

Das POS Tagging und Finden von Korrekturen lieferte nach ersten Einschätzungen brauchbare Ergebnisse. In vielen Artikeln wurden mögliche Korrekturen erkannt.

6.3 NLP Tools und Alternativen

Die ersten Recherchen und Versuche zu dieser Arbeit wurden 2014 durchgeführt. In den vergangenen Jahren hat sich die Sprachverarbeitung deutlich weiterentwickelt. Die großen Cloud Computing Anbieter stellen Tools als Service bereit, welche Maschine Learning und auch Natural Language Processing bieten. Sprachsteuerungen wie Alexa, Siri und Cortana sind beliebt und deren Entwicklungen treiben diese Technologien voran.

Für das *oekoor* Tool fiel die Entscheidung sehr schnell auf das NLTK. Dieses ist vor allem im akademischen Umfeld stark vertreten und bietet hilfreiche Lösungen für die flache Satzverarbeitung. POS Tagging stellte sich anfänglich für die deutsche Sprache als sehr schwierig heraus. Außer dem TIGER Korpus gibt es kaum verfügbare Sammlungen.

Das Tool spaCy [www.spacy.io] kam 2015 auf und wurde evaluiert. Es versprach, viele Open Source Tools zu vereinen und eine viel einfachere Nutzung zu ermöglichen. Neben hoher Performance wurde spaCy auch mit fertigen Modellen ausgeliefert. Hier wurde auch versprochen, welche für Deutsch zu bieten. Das dahinterstehende Unternehmen wurde in Berlin gegründet. Die Entwicklung von spaCy ging aber anfänglich nur langsam voran und *oekorr* wurde deshalb mit NLTK und eigenen Skripten und Anpassungen von Bibliotheken realisiert. Heute könnte man spaCy durchaus als Alternative nutzen.

Nachfolgende Grafik bietet einen Überblick zu verfügbaren Tools. Diese sind mit der Grafik referenziert.

39 Sammlung an verfügbaren NLP Tools (28)

6.4 Zukünftige Entwicklungsmöglichkeiten

Die Evaluierung verschiedenster NLP Tools für *oekorr* ging anfänglich langsam voran und hat viel Zeit beansprucht. Deshalb wurde der Umfang des Prototyps etwas geringer als im Projektplan skizziert.

Oekorr bietet POS Tags, Lemmata und Korrekturvorschläge in GUI und WebAPI an. In weiteren Schritten müsste man diese Korrekturvorschläge wählen können und das Tool sollte diese in einem Autokorrekturmodus mit der POS Information zurück in Fließtext umwandeln. Dieser sollte abgespeichert und heruntergeladen werden können.

Zur Integration in ein Zeitungsredaktionssystem wird man die Schnittstellen auf deren Anforderungen anpassen müssen.

7 Zusammenfassung

In dieser Arbeit wurde eine komplexe Anwendungsentwicklung betrieben, die zum *oekorr* Prototypen geführt hat. Nach ausgiebiger Recherche und unzähligen aufwendigen Experimenten mit vielen Tools und Bibliotheken konnte ein Ansatz gefunden werden, welcher den Anforderungen entspricht und Mechanismen zur automatisierten Korrektur deutscher Texte bietet. Das Zerlegen der Texte sowie die Bestimmung der Wortarten funktioniert mit einer sehr hohen Genauigkeit und kann mit beliebigen Texten demonstriert werden. Das Finden der Lemmata und das darauffolgende Suchen im Ersetzungswörterbuch bietet die Funktion eines Korrekturtools. Mit diesen Vorschlägen ist der Benutzer in der Lage, seine Texte zu korrigieren und den Ansprüchen österreichischer Leser zu genügen. Die Qualität dieser Korrekturen hängt von der Menge der Tautonismen in der Datenbank ab und kann bzw. muss erweitert werden. Das *oekorr* Web Tool bietet hierfür eine Maske.

Mit diesem Prototyp wurde eine solide Basis für eine weitere Entwicklung geschaffen.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

1. **Bundesministerium für Bildung und Frauen.** (Österreichisches) Deutsch. [Online] 2014. https://bildung.bmbwf.gv.at/schulen/unterricht/oed.pdf.

2. **Carstensen, Kai-Uwe, Ebert, Cornelia und Ebert, Christian.** *Computerlinguistik und Sprachtechnologie.* s.l. : Spektrum Akademischer Verlag Heidelberg, 2010. ISBN 978-3-8274-2023-7.

3. **Stuttgart, Universität.** TIGER Corpus. [Online] 2014. http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/tiger.html.

4. **Universität des Saarlandes.** NEGRA Korpus. [Online] 2017. http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/negra-corpus.html.

5. **Thielen, Christine, et al.** *Guidelines für das Tagging deutscher Textcorpora mit STTS.* Stuttgart, Tübingen : s.n., 1999.

6. **Smith, George.** Universität Potsdam. *A Brief Introduction to the TIGER Treebank.* [Online] 2003. http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/TIGERCorpus/annotation/tiger_introduction.pdf.

7. **Albert und Anderssen.** TIGER Annotationsschema. [Online] [Zitat vom: 03. 07 2018.] http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/TIGERCorpus/annotation/tiger_scheme-syntax.pdf.

8. **GermaNet. [Online] Juli 2019. http://www.sfs.uni-tuebingen.de/GermaNet/index.shtml.**

9. Flask's documentation. [Online] 06 2019. https://flask.palletsprojects.com/en/1.1.x/.

10. ***Flask is a microframework for Python.*** **[Online] 09 2018. http://flask.pocoo.org.**

11. Massé, Mark. ***REST API Design Rulebook.*** **s.l. : O'Reilly Media, Inc, 2012. ISBN: 978-1-449-31050-9.**

12. RFC 2617. ***HTTP Authentication: Basic and Digest Access Authentication.*** **[Online] 06 2019. https://tools.ietf.org/html/rfc2617.**

13. RFC 7519. ***JSON Web Token (JWT).*** **[Online] 06 2019. https://tools.ietf.org/html/rfc7519.**

14. Peyrott, Sebastián E. ***The JWT Handbook Version 0.14.1.*** **s.l. : Auth0 Inc, 2016-2018.**

15. Stecky-Efantis, Mikey. 5 Easy Steps to Understanding JSON Web Tokens (JWT). [Online] 2016. https://medium.com/vandium-software/5-easy-steps-to-understanding-json-web-tokens-jwt-1164c0adfcec.

16. MDN web docs. ***Same-origin policy.*** **[Online] mozilla, Juli 2019. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy.**

17. Bird, Steven, Edward Loper and Ewan Klein. ***Natural Language Processing with Python.*** **s.l. : O'Reilly and Associates, 2009. ISBN 978-0596516499.**

18. Freeman, Adam. ***Pro AngularJS.*** **s.l. : Apress, 2014. ISBN 978-1430264484.**

19. Sanderson, Steven. ***Pro ASP.NET MVC Framework.*** **s.l. : Apress, 2009. ISBN 978-1430210078.**

20. KISALAYA PRASAD, AVANTI PATIL, AND HEATHER MILLER. Futures and Promises. [Online] 08 2018. http://dist-prog-book.com/chapter/2/futures.html.

21. Halstead, R. H., Jr. ***MULTILISP: A Language for Concurrent Symbolic Computation.*** **1985.**

22. Pohl, Stephan. Callback-Hölle. ***Web & Mobile Developer.*** **2017, 5.**

23. Nolte, Philipp. ***ClassifierBasedGermanTagger.*** **[Online] Juni 2019. https://github.com/ptnplanet/NLTK-Contributions/blob/master/ClassifierBasedGermanTagger/ClassifierBasedGermanTagger.py.**

24. —. ***NegraCorpusReader.*** **[Online] Juni 2019. https://github.com/ptnplanet/NLTK-Contributions/blob/master/NegraCorpusReader/NegraCorpusReader.py.**

25. Roberts, Will. ***pygermanet.*** **[Online] Juni 2019. https://pypi.org/project/pygermanet/.**

26. ***Bootstrap.*** **[Online] 2018. https://getbootstrap.com.**

27. Klenov, Kirill. py-frameworks-bench. [Online] 07. 08 2019. https://klen.github.io/py-frameworks-bench/#results.

28. 7 Amazing Open Source NLP Tools to Try With Notebooks in 2019. [Online] 08. 08 2019. https://medium.com/microsoftazure/7-amazing-open-source-nlp-tools-to-try-with-notebooks-in-2019-c9eec058d9f1.

29. Hirschmann, Hagen. ***Humboldt-Universität zu Berlin - Institut für deutsche Sprache und Linguistik.*** **[Online] 09 2017. https://www.linguistik.hu-berlin.de/de/institut/professuren/korpuslinguistik/mitarbeiter-innen/hagen/STTS_Tagset_Tiger.**

A. Anhang

STTS-Tags gemäß Tiger-Annotationsschema

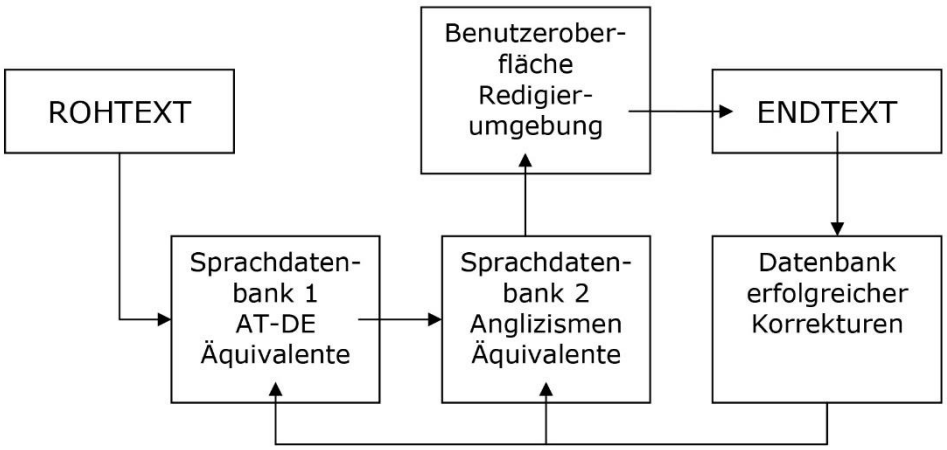
pos-tag	Beschreibung	Beispiel(e)
ADJA	attributives Adjektiv	*der schlaue*/ADJA *Mitarbeiter*
ADJD	adverbiales ODER	*er spricht schnell*/ADJD
	prädikatives Adjektiv	*Sein Sprechen ist schnell*/ADJD
ADV	Adverb	*Bald*/ADV *schon*/ADV *kommt sie wohl*/ADV
APPR	Präposition; Zirkumposition links	*nach*/APPR *Berlin; ohne*/APPR *Hund*
APPRART	Präposition mit Artikel	*zum*/APPRART *Streichen; zur*/APPRART *Sache*
APPO	Postposition	*ihm zuliebe*/APPO*; der Sache wegen*/APPO
APZR	Zirkumposition rechts	*von mir aus*/APZR
ART	bestimmter ODER	*Der*/ART *Mann schenkt die*/ART *Rose*
	unbestimmter Artikel	*einer*/ART *unerwarteten Frau*
CARD	Kardinalzahl	*zwei*/CARD *Männer im Jahre 1994*/CARD
FM	Fremdsprachliches Material	*Er sagte:" Hasta*/FM *luego*/FM*, amigos*/FM *."
ITJ	Interjektion	*Mhm*/ITJ*, ach*/ITJ*, tja*/ITJ*, dann halt nicht.*
KOUI	unterordnende Konjunktion mit (*zu*-)Infinitiv	*Sie kommt, um*/KOUI *zu arbeiten*
		Anstatt/KOUI *anzufangen, geht sie wieder*
KOUS	unterordnende Konjunktion	*Emma wartet, weil/ob/solange/dass*/KOUS *sie stiehlt*
KON	nebenordnende Konjunktion und, oder, aber	*Sie und/oder*/KON *Emma kommen und*/KON *streichen*
KOKOM	Vergleichskonjunktion als, wie	*blauer als*/KOKOM *er; blau wie*/KOKOM *er*
NN	normales Nomen	*am Tage*/NN *dem Mann*/NN *den Schlaf*/NN
NE	Eigennamen	*die Emma*/NE *dem Hans*/NE *sein HSV*/NE
PDAT	attribuierendes Demonstrativpronomen	*Jene*/PDAT *Männer sprachen dieses*/PDAT *lockere Spanisch*
PDS	substituierendes Demonstrativpronomen	*Denen*/PDS *war dies*/PDS *nicht übelzunehmen*
PIAT	attribuierendes Indefinitpronomen	*Manche*/PIAT *Rose währt einige*/PIAT *Tage*
PIS	substituierendes Indefinitpronomen	*Manche*/PIS *verzeiht niemandem*/PIS
PPER	(nicht-reflexives) Personalpronomen	*Er*/PPER *schenkt sie*/PPER *ihr*/PPER
PPOSAT	attribuierendes Possessivpronomen	*Unsere*/PPOSAT *Wand ist rosa*
PPOSS	substituierendes Possessivpronomen	*Meiner*/PPOSS *schlägt deinen*/PPOSS
PRELS	substituierendes Relativpronomen	*die Mannschaft, der*/PRELS *du nacheiferst*
PRELAT	attribuierendes Relativpronomen	*die Mannschaft, deren*/PRELAT *Aura du verehrst*
PRF	Reflexivpronomen	*Erinnere dich*/PRF*, wie er sich*/PRF *ereiferte*
PWS	substituierendes Interrogativpronomen	*Wer*/PWS *hat wen*/PWS *gestohlen?*
PWAT	attribuierendes Interrogativpronomen	*Wessen*/PWAT *Hund wurde gestohlen?*
PWAV	adverbiales Interrogativpronomen	*Warum*/PWAV *schneidest du die Rose?*
PROAV	Pronominaladverb	*Deswegen*/PROAV *sprechen wir darüber*/PROAV
PTKZU	"*zu*" vor Infinitiv	*Ich versuche zu*/PTKZU *verschlafen*
PTKNEG	Negationspartikel *nicht*	*Nicht*/PTKNEG *schlecht, wie du nicht*/PTKNEG *hinsiehst*
PTKVZ	abgetrennter Verbzusatz/Verbpartikel	*Pass auf*/PTKVZ *und hör weg*/PTKVZ*!*
PTKANT	Antwortpartikel	*ja; nein; danke; bitte*
PTKA	Partikel "*am*" o. "*zu*" vor Adjektiv o. Adverb	*Zu*/PTKA *teure Rosen welken am*/PTKA *schnellsten*
TRUNC	abgetrenntes Kompositionserstglied	*Mallorca liegt zwischen An-*/TRUNC *und Abreise*
VVFIN	finites Vollverb	*Wir passen*/VVFIN *auf und hören*/VVFIN
VAFIN	finites Voll- oder Kopulaverb	*Sie ist*/VAFIN *blumig. Du hast*/VAFIN *weggehört.*
VMFIN	finites Modalverb	*Sie sollte*/VMFIN *passen*
VVINF	infinites Vollverb	*Wir wollen weghören*/VVINF.
VAINF	infinites Hilfsverb oder Kopulaverb	*Sie soll rot geworden sein*/VAINF
VMINF	infinites Modalverb	*Er hat nicht schlafen können*/VMFIN.
VVIMP	Vollverb im Imperativ	*Pass*/VVIMP *auf und hör*/VVIMP *weg!*
VAIMP	Kopulaverb im Imperativ	*Sei*/VAIMP *wach!*
VVPP	partizipiales Vollverb (Partizip II)	*Wir haben verschlafen*/VVPP.
VAPP	partizipiales Hilfs-/Kopulaverb (Partizip II)	*Das ist verdrängt worden*/VAPP
VMPP	partizipiales Modalverb (Partizip II)	*Sie hat spielen gedurft*/VVPP
VVIZU	Vollverb/Partikelverb im "*zu*"-Infinitiv	*Wir planen wegzuhören*/VVIZU
XY	Nichtwort, Sonderzeichen, Kürzel	*Es enthält viel D2XW3*/XY
$,	Komma	*,*
$(	sonstige satzinterne Interpunktion	*( )* u.a.
$.	satzbeendende Interpunktion	*. ? ! ;*

ÖKORR: Korrekturprogramm für Medientexte von Nachrichtenagenturen

Kurzbeschreibung
ÖKORR ist Computerprogramm, das in der Lage ist, Texte, die in den Zeitungsredaktionen aus Nachrichtenredaktionen und anderen Quellen eintreffen, dahingehend zu überprüfen, ob der Text dem österreichischen Sprachgebrauch entspricht, inadäquate Ausdrücke/Formulierungen markiert (oder im Auto-Modus) selbständig korrigiert, diese Stellen kennzeichnet, damit sie eventuell noch manuell adaptiert werden können. (Analog zur Korrekturverfolgung in Word und anderen Textverarbeitungsprogrammen).
Das Programm kann auch dahingehend angepasst werden, die Texte in Hinblick auf die Verwendung von Anglizismen bzw. komplexen Wortzusammensetzungen zu überprüfen, die vielen Lesern ein Ärgernis/Leseschwierigkeit sind, und adäquate Alternativausdrücke vorschlagen.
Die Texte werden für die Leser damit vertrauter und einfacher lesbarer. Der Vorteil für Redaktionen besteht darin, dass die Bearbeitungszeit für Texte markant verkürzt und Zeit und Kosten gespart werden.

Struktur des Programms:
Das Programm kann entweder über eine Schnittstelle in ein gegebenes Redaktionssystem eingebunden oder als Stand-alone Variante verwendet werden.
Das Programm wirkt als Filter, durch das die Texte durchgeschleust und korrigiert werden. Es vergleicht die vorhandenen Ausdrücke und Formulierungen anhand der Einträge in den Datenbanken. Anschließend können die Texte von Redakteuren weiter redigiert werden.
Das Programm ist selbst-lernend, d.h., dass die Korrekturen laufend in eine Datenbank aufgenommen werden, wodurch die Korrekturleistung ständig verbessert wird.

Komponenten des Programms



Die Identifizierung der inadäquaten Ausdrücke erfolgt über Einträge in den Datenbanken (1) und (2) sowie über sog. decision trees, Gaussian mixture Modelle, Hidden Markov Modelle, Maximum Entropy Markov Modelle, and andere bewährte stochastische Verfahren, deren Treffsicherheit durch die Eingabe erfolgreicher Korrekturen laufend verbessert wird.
Ersteller: TU-GRAZ / Forschungszentrum UNI Graz
Kontakt: Prof. Rudolf Muhr (0699 10 500 537) / Maximillian Schafzahl / Prof. Denis Helic (0316873 9280) (TU GRAZ)

Academic Research License Agreement

GERMANET DATA ACADEMIC RESEARCH LICENSE AGREEMENT

Between:

Institute: FORSCHUNGSSTELLE ÖSTERREICHISCHES DEUTSCH

Address: Heinrichstraße 22/2 8010 Graz, Österreich

Represented by: Ass. Prof. Dr. Rudolf Muhr

duly authorized to sign this Agreement,

hereinafter referred to as the LICENSEE of GERMANET DATA,

and

Seminar für Sprachwissenschaft

Abt. Allgemeine Sprachwissenschaft und Computerlinguistik

Eberhard-Karls-Universität Tübingen

Wilhelmstr. 19

D-72074 Tübingen

Tel. +49 (0)7071 29 74279

Fax. +49 (0)7071 29 5214

represented by Prof. Dr. Erhard W. Hinrichs, Director,

hereinafter referred to as the LICENSOR of GERMANET DATA,

the following Agreement is made, and shall be effective as of Date: July 08 2015
(Month, Day, Year)

LICENSE CONDITIONS

DEFINITIONS

a. "Agreement" shall mean this entire Agreement, which is composed of the 19 clauses herein together with APPENDIX A and APPENDIX B thereafter.
b. "GERMANET DATA" as described in APPENDIX A, are the results of the SLD project, funded by the Ministerium für Forschung, Bildung und Kunst Baden-Württemberg. The GERMANET DATA are property of the LICENSOR, the Seminar für Sprachwissenschaft, Abt. Computerlinguistik, represented by Prof. Erhard Hinrichs, as legal copyright holder of GERMANET.
c. "Software" shall mean proprietary GERMANET DATA software, including all versions, updates, releases, derivative works, modifications and documentation relating thereto or thereof that may be provided by LICENSOR to LICENSEE.
d. "Derived product or service" shall mean any derivative work that is based on the GERMANET DATA, such as revision, enhancement, modification, translation, localization, abridgement, condensation, expansion, or any other form into which the GERMANET DATA may be transformed or that, if prepared without the authorization of the owner of the copyright in such pre-existing work, would constitute a copyright infringement.

GRANT

1. The LICENSOR of the GERMANET DATA hereby grants to LICENSEE a non-exclusive ACADEMIC RESEARCH LICENSE, subject to the terms of this Agreement.

ACADEMIC RESEARCH LICENSE

2. LICENSOR grants LICENSEE a non-exclusive non-transferable right to use the GERMANET DATA for the purpose of non-commercial, non-profit research by LICENSEE for any research project conducted on the premises of LICENSEE. LICENSEE agrees to inform LICENSOR in written form on a yearly basis about the academic research projects for which the GERMANET DATA are currently utilized. LICENSEE is not allowed to distribute and market any derived product or service based on the GERMANET DATA. The commercial use of the GERMANET DATA shall require a separate Agreement with LICENSOR. If LICENSEE intends to use the GERMANET DATA for academic purposes other than research (e.g. teaching), LICENSEE shall inform LICENSOR in advance of such uses in order to obtain non-disclosure agreements to be signed by any third parties who may gain access to GERMANET DATA.

OBLIGATIONS

3. LICENSEE undertakes to treat as confidential and keep secret any information about or related to the software conveyed to the LICENSEE by LICENSOR.
4. LICENSEE undertakes to respect LICENSOR's intellectual property rights with regard to the GERMANET DATA. These include all trademarks, trade names, good will associated with trademarks and trade names, copyrights, patents, trade secrets, know-how, rights of confidentiality and confidence, and any other intellectual property rights and all applications for any of the above or any registration thereof.

5. The GERMANET DATA shall neither be assigned nor sub-licensed to any other third party or joint venture partner by the LICENSEE.
6. LICENSEE grants LICENSOR the right to publicly disclose the existence of this Agreement in LICENSOR's public documentation, including but not limited to information on the GermaNet website or in GermaNet leaflets.
7. The linguistic content of the GERMANET DATA shall not be altered, modified or changed by the LICENSEE without prior written LICENSOR's consent and agreement.
8. LICENSEE agrees to acknowledge the use of the GERMANET DATA and associated resources in all publications reporting on results obtained with the help of the GERMANET DATA and to give appropriate references to LICENSOR in scholarly literature when the GERMANET DATA are mentioned. LICENSEE agrees to cite GermaNet and its associated resources according to the specifications summarized in APPENDIX B.
9. LICENSEE shall ensure that no persons or institutions other than persons that are in the employ of LICENSEE or that are members of a research project conducted on the premises of LICENSEE will get access to the GERMANET DATA or parts thereof. Other persons or institutions seeking to use the GERMANET DATA for research purposes or commercial purposes should be directed to LICENSOR for license agreements.
10. LICENSEE shall not, without the authorization of LICENSOR, make available to the public all or a substantial part of the contents of the GERMANET DATA, evaluated quantitatively and/or qualitatively, by the distribution of copies, by renting, leasing or any other form of distribution.
11. LICENSEE shall not, without the authorization of LICENSOR, make the GERMANET DATA available to third parties in any form that allows to reconstruct the original data.
12. GERMANET DATA are provided to LICENSEE "as is", without any warranty and without any guarantee of its usability. LICENSOR excludes all liability of whatsoever nature for direct, consequential or indirect loss or damage suffered by the LICENSEE in connection with the distribution of GERMANET DATA. LICENSOR MAKES NO REPRESENTATIONS OR WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE LICENSED SOFTWARE, DATABASE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

DISTRIBUTION

13. GERMANET DATA will be distributed within thirty (30) days after the effective date as an archive attached to an email or any other medium the partners agree upon.

TERMINATION

14. Either LICENSOR or LICENSEE may terminate this LICENSE at any time by notifying the other party in writing thirty (30) days before the date of termination. In addition, LICENSOR may terminate this LICENSE immediately upon LICENSEE's breach of conditions by notifying LICENSEE of termination.

RIGHTS

15. All rights not specifically granted in this LICENSE are reserved by LICENSOR.
16. LICENSOR confirms that LICENSEE is exempt from obligations that originate from the rights of third parties.
17. Neither party shall be held responsible for any delay or failure to perform any of its obligations caused by "force majeure". Should such an event occur, the party experiencing such delay is obliged to inform the other party thereof.

MODIFICATION OF LICENSE

18. This LICENSE can only be modified by further formal Agreement in writing between LICENSOR and LICENSEE.

DISPUTES

19. This LICENSE shall be governed by German Law. Place of jurisdiction is Tübingen.

The entire Agreement is composed of the 19 clauses herein together with Appendix A and Appendix B thereafter.

In witness whereof, intending to be bound, the parties here to have hereunder set their hands and executed this LICENSE by their duly authorized representatives:

AUTHORIZED SIGNATURES :

On behalf of **LICENSEE**	On behalf of **LICENSOR**
Name: Rudolf Muhr	Name: Erhard W. Hinrichs
Title: Ass. Prof. Dr.	Title: Professor and Director
Date: July 08 2015	Date:

APPENDIX A: **GERMANET DATA DESCRIPTION**:

GermaNet is an online lexical-semantic database for German that has been structured along the same lines as the Princeton WordNet for English. It models nouns, verbs, and adjectives in their basic semantic relations like hyponymy, meronymy, and antonymy, and constitutes a basic resource for word sense disambiguation within NLP applications.

The development of the GermaNet resource has been directed by Prof. Erhard Hinrichs and has been supported by research grants awarded to Prof. Hinrichs by the Ministerium für Wissenschaft und Kunst Baden-Württemberg (MWK), the Bundesministerium für Bildung und Forschung (BMBF), the Deutsche Forschungsgemeinschaft (DFG), and the European Commission. The following researchers and research assistants (listed in alphabetic order) have contributed to the construction of GermaNet and accompanying software:

Reinhild Barkey, Agnia Barsukova, Valérie Béchet-Tsarnos, Anne Brock, Julia Chant, Edo Collins, Bettina Demmer, Valentin Deyringer, Sabrina Galasso, Piklu Gupta, Annabell Grasse, Helmut Feldweg, Patricia Fischer, Clemens Frey, Marina Haid, Birgit Hamp, Verena Henrich, Marie Hinrichs, Michael Hipp, Christina Hoppermann, Lars Horber, Silke Kugler, Claudia Kunze, Anja Laske, Lothar Lemnitzer, Andrea Lorenz, Karin Naumann, Till Pachalli, Katrin Petodnig, Niko Schenk, Susanne Schüle, Sarah Schulz, Daniil Sorokin, Rosemary Stegmann, Daniela Stelle, Daniela Stier, Steffen Tacke, Christine Thielen, Tatiana Vodolazova, Andreas Wagner, Johannes Wahle, Zarah-Leonie Weiss, Martin Wolf and Holger Wunsch.

The **GERMANET DATA** consist of the following modules: 54 GermaNet object files, 1 GermaNet relations file, 1 file containing the Interlingual Index, 3 files containing the GermaNet-Wiktionary mapping (including the Wiktionary sense descriptions that extend GermaNet's lexical units), 4 DTDs describing the format of these files, 1 database dump file, and 2 documentary files (a general README and a database setup description).

These files provide the following pieces of information for each synset:

- set of variants or synonyms making up the synset as basic unit of the wordnet
- part-of-speech
- semantic relations to other synsets
- spelling variants (optional)
- usage labels (optional)
- glosses (optional)
- examples (optional)
- verb frames (optional)

GermaNet is still under development, and its coverage is constantly being extended. The current size of the version 10.0 database is:

101371	Synsets
131814	Lexical units
119231	Literals
1.3	Lexical units per synset
114619	Number of conceptual relations
4199	Number of lexical relations (synonymy excluded)
54572	Number of split compounds
28567	Number of Interlingual Index records
29535	Number of Wiktionary sense descriptions

APPENDIX B: **CITING GERMANET AND ASSOCIATED RESOURCES**:

To refer to the **GermaNet resource** in publications cite **both** of the following two papers:

Hamp, Birgit and Helmut Feldweg. "GermaNet - a Lexical-Semantic Net for German." *Proceedings of the ACL workshop Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications.* Madrid, 1997.

Henrich, Verena and Erhard Hinrichs. "GernEdiT - The GermaNet Editing Tool". *Proceedings of the Seventh Conference on International Language Resources and Evaluation (LREC 2010).* Valletta, Malta, May 2010, pp. 2228-2235.

To refer to the **nominal compounds** included in GermaNet in publications cite the following paper:

Henrich, Verena and Erhard Hinrichs. "Determining Immediate Constituents of Compounds in GermaNet." *Proceedings of Recent Advances in Natural Language Processing (RANLP 2011).* Hissar, Bulgaria, Sept 2011, pp 420-426.

To refer to the **GermaNet-Wiktionary mapping** and the harvested definitions in the context of scientific or research work cite the following paper:

Henrich, Verena, Erhard Hinrichs, and Tatiana Vodolazova. "Aligning GermaNet Senses with Wiktionary Sense Definitions." *Human Language Technology: Challenges for Computer Science and Linguistics.* Eds. Zygmunt Vetulani and Joseph Mariani. *Lecture Notes in Computer Science.* Vol. 8387, 2014, pp. 329-342.

You can find download links to the various papers on our website:
http://www.sfs.uni-tuebingen.de/GermaNet/citing_germanet.shtml