Jeton Arifi, Bsc

Potentiale von Chatbots für die Rechtschreibprüfung im Kindesalter

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

Masterstudium Softwareentwicklung – Wirtschaft

eingereicht an der

Technischen Universität Graz

Betreuer:

Priv.-Doz. Dipl.-Ing. Dr.techn. Martin Ebner

Mitbetreuer:

Dipl.-Ing. Markus Ebner

Institute of Interactive Systems and Data Science

25. April 2019

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 entnommene Stellen als solche kenntlich gemacht habe.

Graz, am …… ……
(Unterschrift)

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, am …… ……
(Unterschrift)

ABSTRACT

Within the scope of this Masterthesis a prototype of a Chatbot has been developed that shall be integrated within the IDeR-Blog Learning platform of the Graz University of Technology. The IDeRChatbot should act as a tutor for spelling mistakes and offer it's users a human-like communication. In addition, children should be incited to find solutions by typing questions (text inputs) to the IDeRChatbot.

Therefore, a LanguageTool API for correction of spelling errors was integrated within the IDeRChatbot. Additionally, an algorithm was developed in the Backend that corrects orthographic words. Through this algorithm the IDeRChatbot has the ability to learn because the correction suggestions become more precise over time through the feedback of the users.

In this master thesis the topic was introduced by the inclusion of the technical literature and afterwards the IDeRBlog-Chatbot with the used technologies and functions was described. Likewise, the implementation of the prototype was discussed in detail and presented comprehensibly with illustrations and listings. Finally, results of the testing were explained.

Keywords: Chatbot, IDeRBlog, IDeRBlog-Chatbot, learning app, spelling, misspelling correction, learning platform

KURZFASSUNG

Im Rahmen dieser Masterarbeit wurde ein Prototyp eines Chatbots (IDeRBlog-Chatbot) für die IDeRBlog Lernplattform der Technischen Universität Graz entwickelt. Der Prototyp soll später in diese Lernplattform integriert werden. Der IDeRBlog-Chatbot soll als Tutor für Rechtschreibfehler fungieren und soll den BenutzerInnen eine menschenähnliche Kommunikation bieten. Dabei sollen die Kinder auch animiert werden, Lösungen durch Fragestellungen (Texteingaben) an den IDeRBlog-Chatbot zu finden.

In den IDeRBlog-Chatbot wurde daher das LanguageTool API für die Korrektur der Rechtschreibfehler integriert. Zusätzlich wurde im Backend ein Algorithmus zur Korrektur von orthografischen Rechtschreibfehler entwickelt. Durch diesen Algorithmus ist der IDeRBlog-Chatbot auch lernfähig, weil die Korrekturvorschläge durch die Anwendung und das Feedback von den NutzerInnen präziser werden.

In dieser Masterarbeit wurde durch den Einbezug der Fachliteratur an die Thematik herangeführt und es wurden im Anschluss der IDeRBlog-Chatbot mit den eingesetzten Technologien und Funktionen beschrieben. Ebenso wurde die Umsetzung des Prototypen ausführlich thematisiert und mit Abbildungen und Auflistungen nachvollziehbar dargestellt. Abschließend wurden Ergebnisse der Testung erläutert.

Schlagwörter: Chatbot, IDeRBlog, IDeRBlog-Chatbot, Lernplattform, Learning-App, Rechtschreibung, Rechtschreibfehlerkorrektur

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

AUFLISTUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

AIML	Artificial Intelligence Markup Language
ALICE	Artificial Linguistic Internet Computer Entity
AOP	Aspektorientierte Programmierung
API	Application Programming Interface (Anwendungsschnittstelle)
CSS	Cascading Style Sheets
DI	Dependency Injection
DIE	Integrated Development Environment
GUI	Graphic User Interface
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IDeR	Individuell Differenziert Richtig Schreiben
JSON	JavaScript Object Notation
KI	Künstliche Intelligenz
MIT	Massachusetts Institute of Technology
MUD	Multi User Dialog
PC	Personal Computer
POM	Project Object Model
TU	Technische Universität
URL	Uniform Resource Locator
US	United States
WAR	Web Application Resource
XML	Extensible Markup Language
z.B.	zum Beispiel

1 EINLEITUNG

1.1 Motivation

IDeRBlog ist ein Projekt, welches die Technische Universität Graz in Kooperation mit verschiedenen Bildungseinrichtungen durchführt. IDeRBlog ist eine Internetplattform bzw. ein digitales Medium, welche eine Übungsdatenbank mit zahlreichen kostenlosen Übungen umfasst, die unter einer Creative Commons-Lizenz lizenziert ist. Die online verfassten Blogeinträge oder Texte von SchülerInnen der Grundschulen werden durch ein intelligentes Wörterbuch auf ihre Rechtschreibung geprüft und es werden Korrekturvorschläge angeboten. Auf Basis dieser Rechtschreibfehler werden Rechtschreibkorrekturen den VerfasserInnen bzw. SchülerInnen vorgeschlagen. Ebenso können Lehrkräfte auf diese Rechtschreib-fehleranalyse zugreifen, um diese im Unterricht weiter zu verwenden. Derzeit wird eine Ausweitung der Plattform durch einen Chatbot angestrebt, um noch mehr Motivation bzw. eine verbesserte Fehleranalyse den BenutzerInnen zu bieten.

1.2 Problemstellung

Die aktuelle Version der IDeRBlog-Lernplattform hilft Kindern Rechtschreibfehler in Texten zu korrigieren. Während der Analyse wurde herausgefunden, dass die Plattform für orthografische Begriffe, Umgangssprache und unterschiedliche Sprachvariationen nicht korrekte

Vorschläge liefert. Der entwickelte IDeRBlog-Chatbot soll für diese Fehler besser Korrekturvorschläge liefern und gleichzeitig eine einfache Interaktion mit den BenutzerInnen ermöglichen, die sie auch zum Lernen anregt.

1.3 Aufgabe

Die Aufgabenstellung im Rahmen dieser Masterarbeit umfasst die Entwicklung und die Programmierung eines Chatbot-Prototypens, der in die IDeRBlog Lernplattform integriert wird. Die Zielgruppe umfasst SchülerInnen im Kindesalter, die die deutsche Sprache erwerben. Dieser Chatbot sollte beim Verfassen der Texte und Einträge auf die Rechtschreibfehler, die die VerfasserInnen machen, aufmerksam machen und Verbesserungsvorschläge bieten.

Die SchülerInnen sollen durch Eingaben und Fragestellungen mit dem Chatbot interagieren. Beispielsweise durch eine Fragestellung, wie man bestimmte Wörter schreibt. Die Interaktion, Eingabe und Bedienung des Chatbots soll für die BenutzerInnen sehr einfach gestaltet sein. Den SchülerInnen sollte auch das Gefühl gegeben werden, wie mit einem Betreuer bzw. Tutor kommunizieren zu können, der sie bei ihren Lernprozessen bestmöglich unterstützt.

1.4 Aufbau der Arbeit

Die vorliegende Masterarbeit umfasst theoretische Grundlagen sowie die ausführliche Beschreibung der praktischen Entwicklung eines Prototypen bzw. IDeRBlog-Chatbots. Die theoretischen Inhalte befassen sich mit der Auseinandersetzung mit der Fachliteratur und mit der Heranführung an die Thematik.

Das erste Kapitel „Allgemeines" (Kapitel 2) ist der Thematik Chatbot gewidmet. Es wird die Definition „Chatbot" erläutert und im Anschluss werden

der Loebner Wettbewerb und der Turing Test, welcher als Qualitätsstandard angesehen wird, beschrieben. Die historischen Entwicklungsschritte und die namhaftesten Chatbots, die im Laufe der Zeit entwickelt worden sind, werden im Abschnitt 2.3 angeführt. Die Funktionsweisen von Chatbots und deren menschenähnliche Züge werden im Abschnitt 2.4 und 2.5 behandelt. Im Abschnitt 2.6 werden Unterscheidungs- bzw. Klassifizierungsmerkmale aufgelistet und abschließend werden unterschiedliche Gebiete, in denen Chatbots bereits erfolgreich eingesetzt werden beschrieben. Das Kapitel 3 widmet sich dem Prototypen des IDeRBlog-Chatbots. Die Aufgabenstellung wird zu Beginn im Abschnitt 3.1 angeführt und näher erläutert. Abschnitt 3.2 widmet sich den eingesetzten Technologien und der Entwicklungsumgebung. In diesem Abschnitt werden die verwendeten Technologien HTML, CSS, JavaScript, Websocket, Bootstrap und der Java Spring Boot beschrieben. Es wird ebenso auf die eingesetzte Entwicklungsumgebung IntelliJ näher eingegangen. Die Levenshtein Distanz-Funktion, das LanguageTool und Lombok werden im darauffolgenden Abschnitt 3.3 behandelt. Die Umsetzung des Prototypen des IDeRBlog-Chatbots wird im Kapitel 4 umfassend beschrieben. Abschnitt 4.1 widmet sich der Chatbot-Architektur, die auf einer Client-Server-Architektur basiert und im folgenden Abschnitt 4.2 wird die Implementation des Clients bzw. des Webfrontends aufgezeigt. Das Design bzw. die Bedienoberfläche des IDeRBlog-Chatbots wird mit Abbildungen anschaulich dargestellt, um einen Einblick in das Aussehen des IDeRBlog-Chatbots zu geben. Des Weiteren wird die Implementation vom Backend, inklusive der Konfiguration, der Websocket-Verbindung, der Definition der Absichten (Itents), der Extrahierung von Schreibfehlern, Korrekturvorschlägen und im Anschluss die Generierung der Antwort, beschrieben. Für eine bessere Nachvollziehbarkeit wurden verschiedene Abbildungen in diesem Abschnitt hinzugezogen. Die Testung des Prototypens wurde im Kapitel 5 kurz beschrieben. Diese vorliegende Masterarbeit schließt mit einer Zusammenfassung (Kapitel 6) ab, in der die wichtigsten Punkte nochmals zusammengefasst werden.

2 ALLGEMEINES

In diesem Kapitel wird ein allgemeiner Überblick über die Thematik Chatbot gegeben. Im Abschnitt 2.1 wird die Begriffsdefinition Chatbot näher erläutert und im Kapitel 2.2 wird der Loebner Preis, bei dem das menschenähnlichste Programm in einem abgewandelten Turing Test ermittelt wird, beschrieben. Der Abschnitt 2.3 widmet sich den historischen Entwicklungsschritten bei der Entwicklung von Chatbots und es werden die EntwicklerInnen und die bedeutendsten Chatbots, die im Laufe der Zeit erfolgreich programmiert wurden, kurz angeführt. Die Funktionsweise von Chatbots, vor allem der wichtige Zugriff auf die Datenbank, wird im Kapitel 2.4 erläutert und es wird auf die Maßnahmen, die ergriffen werden, um Chatbots menschenähnlicher zu gestalten eingegangen. Im darauffolgenden Kapitel 2.5 werden menschlichen Züge eines Chatbots aufgezeigt. Ebenso werden tabellarisch die Klassifizierungsmerkmale im Kapitel 2.6 und verschiedene Einsatzgebiete und -möglichkeiten von Chatbots im Kapitel 2.7 beschrieben.

2.1 Definition Chatbot

Der Begriff „Chatbot" setzt sich aus dem englischen Wort „Chat" (Gespräch) und „bot" (Abkürzung von „robot") zusammen. Unter einem Chatbot wird ein Computersystem verstanden, welches dem Menschen eine auf natürliche Sprache basierende Interaktion mit dem Computer, die über Tastatur oder Stimmerkennung erfolgt, ermöglicht. Ein Chatbot arbeitet weitgehend nach sich wiederholenden Mustern ohne menschliches Zutun und simuliert Ansprechpartner. Dabei greifen Chatbots auf eine hinterlegte Wissensbasis

(Knowledge-Datenbank) zu, in der sie durch das Aufspüren von Übereinstimmungen gestellter Fragen mit dem vorhandenen, die vorher von Programmierern erstellten Fragenbestand zugehörige Aktionen bzw. Antworten auswählen und dem Fragenden als Antwort ausgeben.

Chatbots sind teilweise personifiziert, das heißt sie werden zum Teil in Verbindung mit einem Avatar (Mensch, Tier oder Fabelwesen), oftmals als reine Texteingabeboxen, benutzt. Mit der Hilfe dieses „lebenden Organismus" und seiner Natürlichkeit soll der Umgang mit dem Computer unterhaltsam und mühelos werden (vgl. Braun 2003, S.21; Willmes 2014, o. S.). Ebenso können Chatbots nützliche Dienste anbieten, auf wichtige Informationen hinweisen und den Nutzern relevante Internetseiten zusätzlich zu der direkten Beantwortung der gestellten Frage präsentieren. So finden die BenutzerInnen die gesuchten Informationen mit wenig Aufwand bzw. kann das Gespräch durch den Chatbot in eine bestimmte Richtung geführt werden (vgl. Möbus et al. 2006, S.83).

Chatbots übernehmen auch andere Aufgaben neben dem reinen Chat wie beispielsweise Kundenservice in Online-Shops, die Verwaltung von Foren, Internet-Auktionen, Informationssuche im World Wide Web oder stellen künstliche Charaktere in MUD`s (in Deutsch: Multi User Dialog) dar. Auch Internetauftritte von Firmen werden immer mehr mit einem Chatbot ergänzt, der als virtueller Gastgeber fungiert, Fragen zu Produkten und Dienstleistungen beantwortet und durch die Webseite führt. Dadurch erhoffen sich die Firmen eine stärkere Kundenbindung und durch die Gesprächsprotokollauswertung bessere Marketinginformationen.

Außerdem existiert eine Reihe von nicht-kommerzieller Chatbots. Entwickelt werden dies in erster Linie aus Spaß am Experimentieren, nicht zuletzt in der Hoffnung, ein wirklich intelligentes Programm zu schaffen. Jährlich treffen sich überwiegend die EntwicklerInnen dieser nicht-kommerziellen Chatbots zum Wettbewerb um den Loebner Preis, bei dem das menschenähnlichste Programm in einem abgewandelten Turing Test ermittelt wird (vgl. Storp

2002, S.4). Der nächste Abschnitt beschäftigt sich daher auch mit dem Turing Test, welcher als Qualitätsstandard eines Chatbots angesehen wird.

2.2 Loebner Wettbewerb und Turing Test

Der Loebner Wettbewerb wurde vom Soziologen Hugh Loebner im Jahr 1990 initiiert und wurde im Jahr 1991 zum ersten Mal durchgeführt (vgl. Möbus et al. 2006, S.81). Jährlich konkurrieren in diesem Wettbewerb die Chatbots um die Auszeichnung „menschenähnlichstes" Programm. Es ist jedoch das Fernziel, den nach dem gleichnamigen englischen Mathematiker benannten Turing Test zu bestehen. Eine Unterscheidbarkeit von Chatbot und Mensch wäre damit nicht mehr gegeben. Somit wäre den Chatbot menschenähnliches, intelligentes Sprachverhalten zu unterstellen).

Mit der Frage „Können Maschinen denken?" leitete der britische Mathematiker Alan M. Turing seinen im Jahr 1950 in der Zeitschrift „Mind" erschienenen Aufsatz „Computing machinery and intelligence" ein (vgl. Storp 2002, S.5; Deshpande et al. 2017, o. S.). Daher kann die Geschichte der Chatbots auf das Jahr 1950 zurückgeführt werden. In diesem bereits genannten Aufsatz beschrieb Turing die Grundlagen von künstlicher Intelligenz und schlug vor, den Turing Test als Maßstab für die Bewertung der Intelligenz eines Computersystems zu sehen (Deshpande et al. 2017, o. S.). Die eingangs gestellte Frage nach der Denkfähigkeit eines Computerprogramms wurde schließlich durch die Frage nach der Kommunikationsfähigkeit eines Computerprogramms von Turing ersetzt. Man müsste der Maschine Intelligenz unterstellen, falls es einem Computerprogramm tatsächlich gelingt die Kommunikationsfähigkeit eines Menschen in natürlicher Sprache zu imitieren. Voraussetzung dafür ist aber, dass es dem Computer gelänge in unabhängigen Versuchen und dauerhaft zu täuschen.

Die Fähigkeit mit natürlicher Sprache umzugehen ist eng verknüpft mit Intelligenz und Denken. Demzufolge wurde der Turing Test schließlich zum Intelligenz-Test. Seit der Veröffentlichung ist die Diskussion um den Turing

Test nicht abgerissen. Einige sehen in ihm den Ausgangspunkt der künstlichen Intelligenz (KI) an, während andere ihn als Intelligenztest für völlig unzureichend und irreführend halten. In seinem Aufsatz hat Turing selbst zahlreiche Einwände mathematischer, philosophischer, religiöser und biologisch-psychologischer Art gegen seine Idee von der denkenden Maschine akzeptiert und zu wiederlegen versucht.

Auch heute, 50 Jahre nach Turing, gibt es in der KI-Forschung keine Einigung über die Aussagekraft des Turing Tests, aber er scheint den Chatbot-EntwicklerInnen als maßgebliche Orientierung zu dienen. Es ist ihr Ziel, ein System zu entwickeln, das den Turing Test besteht. Im „Gespräch" sollte der Chatbot nicht mehr vom Menschen zu unterscheiden sein und er muss einen menschlichen Gesprächspartner perfekt imitieren. In seiner von Turing ursprünglichen Form wurde der Turing Test nie realisiert. Erst gegen Ende der 80-iger Jahre als ausreichend leistungsfähige Computer entwickelt wurden, konnte mit praktischen Versuchen ernsthaft begonnen werden.

Seit 1990, als der Amerikaner Hugh Loebner die „Loebner Prize Competition in Artificial Intelligence" ins Leben rief, wird jährlich ein modifizierter Turing Test durchgeführt, dem sich verschiedene Chatbot-Programme stellen. Der erste Wettbewerb fand im Boston Computer Museum im Jahr 1991 statt, folglich wurde auch erstmals der Turing Test durchgeführt (vgl. Storp 2002, S.5f.).

In den ersten Wettbewerben beschränkte Loebner die Gesprächsthemen und verlangte von den Test-Gesprächspartnern, die gleichzeitig Juroren waren, eine entsprechende Einhaltung des vorgegebenen Themas. Daher sind die ersten Tests als eingeschränkte Turing Tests zu betrachten (vgl. Storp 2002, S.5f.; vgl. Möbus et al. 2006, S.81). Es fiel dennoch den Juroren nicht schwer, die Programme von menschlicher Kommunikation zu unterscheiden. Zur stark eingeschränkten Testvariante hatte Turing selbst angemerkt, dass sie keine Schlüsse auf die Intelligenz des Systems zulässt. Die thematische Einschränkung ist seit 1995 aufgehoben (vgl. Storp 2002, S.8). In den heutigen Tests werden die Themen von den Programmen vorgegeben und

der Juror darf das Thema auch wechseln. Der Juror darf sich zusätzlich mit jedem Chatbot beliebig oft und beliebig lange unterhalten.

Für den Wettbewerb wurden verschiedene Preisgelder ausgesetzt. Für das erste, nicht vom Menschen unterscheidbare Programm im Turing Test wird, eine Goldmedaille verliehen und es ist ein Preisgeld von 100.000 US-Dollar vorgesehen. Um diesen Preis zu gewinnen, muss das Programm mit audiovisueller Eingabe (Erkennung von gesprochener Sprache, Mimik, Gestik, Emotionen usw.) umgehen können und noch dazu die Jurymitglieder überzeugen (vgl. Möbus et al. 2006, S.81). Die Interviewer haben auch die Hauptaufgabe, die teilnehmenden Systeme entsprechend ihrer „Menschlichkeit" in eine Rangfolge zu bringen. Zu bestimmen ist auch der Punkt, an dem ein Programm vom Menschen nicht mehr zu unterscheiden ist. Diese Variante des Turing Tests ist noch keinem Programm bisher gelungen.

Auch wurde ein weiteres Etappenziel, welchen im Jahr 1999 eingeführt wurde, von keinem Programm erreicht. Eine Silbermedaille und 25.000 US-Dollar bekommt jener Wettbewerber, wenn sein Programm eine „Irreführungsquote" von 50% erreicht (vgl. Storp 2002, S.8). Das Programm sollte drei von zehn Prüfern täuschen können und der Wettbewerber, dessen Programm am „menschenähnlichsten" auf die Juroren wirkt, gewinnt eine Bronzemedaille (vgl. Möbus et al. 2006, S.81).

Bisher wurden nur Bronzemedaillen vergeben, obwohl die Leistungsfähigkeit der teilnehmenden Programme sich seit 1991 erheblich verbessert wurden (vgl. Storp 2002, S.9). Jährlich unterziehen sich fünf bis zehn Programme diesen Wettbewerb. In den letzten Jahren trat dabei immer wieder das ALICE-Projekt mit Richard Wallace an und schnitt in den Jahren 2000, 2001 und 2004 am besten ab. Bisher lag keiner der Juroren bei der Einschätzung zwischen Menschen und Programm falsch, sodass nur Bronzemedaillen vergeben wurden.

Der Turing-Test ist genauso umstritten wie der Loebner Preis. Als Ziel seines Wettbewerbs sieht Loebner selbst die Förderung der KI-Forschung, jedoch steht die Mehrheit der Wissenschaftler ihm und seinen Test als kritisch gegenüber (vgl. Möbus et al. 2006, S.82). Wie sich Chatbots in der Zeit entwickelt haben, wird im nächsten Kapitel aufgezeigt und es werden exemplarisch die erfolgreichsten Chatbots und deren EntwicklerInnen kurz beschrieben.

2.3 Historische Entwicklung

Die ersten Chatbots waren jedoch nicht wirklich intelligent, sondern waren Programme mit einer Sammlung von vordefinierten Antworten, die bestimmten Eingaben entsprachen. Sie waren rudimentär und verwendeten Musterabgleiche und eine Zeichenfolgenverarbeitung, um die Unterhaltung zwischen dem Computer und Menschen aufrecht zu erhalten. Sie erzeugten lediglich eine Illusion von der Intelligenz des Computers, aber die Realität war, dass die Programme ein minimales bis kein Kontextverständnis hatten (vgl. Deshpande et al. 2017, o. S.).

2.3.1 ELIZA

Joseph Weizenbaum entwickelte 1964-1966 am Massachusetts Institute of Technology (MIT) ein Programm, ELIZA, das erstmals die Mensch-Maschine-Kommunikation in natürlicher Sprache ermöglichte. ELIZA gilt heute als der erste Chatbot, obwohl Weizenbaum mit diesem System nicht auf den Turing-Test zielte. Weizenbaum simuliert mit ELIZA PsychotherapeutenInnen in der Gesprächstherapie, da er sie für einfach umzusetzen hielt. Im Wesentlichen beruht nämlich die Aktivität der PsychotherapeutenInnen darauf, die Aussagen der PatientenInnen als Frage zu paraphrasieren, sie zu „spiegeln" und so die PatientenInnen zum Weiterreden zu motivieren. Die

TherapeutInnen reagierten nur auf Gesagtes und verzichten vollkommen auf eigene Gedächtnisinitiativen (vgl. Storp 2002, S.9).

Zunächst untersuchte ELIZA den von BenutzerInnen eingegebenen Text und prüfte ihn auf bestimmte Schlüsselwörter. Anschließend bewertete die diese und wandelte die Eingabe in eine Antwort um. Das Skript, welches ELIZA ausführt, legte die Schlüsselwörter fest, bewertete die Schlüsselwörter und setzte die Transformationsregeln für die Ausgabe fest (vgl. Deshpande et al. 2017, o. S.). Auf dem Bildschirm wurde letztendlich die so erzeugte Antwort ausgegeben.

Abbildung 1: Konversation mit dem ELIZA-Chatbot (Mathur Amit, 2002)

1966 wurde daher ELIZA euphorisch aufgenommen, obwohl Weizenbaum hingegen sein Programm für überbewertet hielt. In ELIZA sahen viele PsychologenInnen die Geburtsstunde der automatisierten Therapie und in KI-Kreisen wurde das Programm als Durchbruch beim maschinellen Verstehen natürlicher Sprache betrachtet. Durch die weitere KI-Forschung wurde aber diese Einschätzung relativiert, da sich die Verarbeitung von natürlicher Sprache als eine sehr große Herausforderung darstellt.

Auch heute noch existiert ELIZA in zahlreichen Versionen im Internet und kann sich auch noch heute mit einigen der modernen Chatbots messen. Als direkte Weiterentwicklung können auch viele moderne Chatbots betrachtet werden (vgl. Storp 2002, S.11).

2.3.2 PARRY

PARRY wurde 1972 vom Psychiater Kenneth Colby an der Stanford University geschrieben. Dieser Chatbot versuchte, eine Person mit paranoider Schizophrenie zu simulieren. PARRY verkörperte eine Konversationsstrategie und war somit ein viel fortschrittlicheres Programm als ELIZA. Dieser Chatbot war als „ELIZA mit bestimmten Eigenschaften bzw. einer bestimmten Haltung" beschrieben worden. PARRY wurde Anfang der 70er Jahre mit einer Veränderten Form des Turing-Test getestet (vgl. Deshpande et al. 2017, o. S.) und trat gegen menschliche Paranoiker an. Dabei hatten selbst erfahrende Therapeuten Schwierigkeiten, Mensch und Computer zu unterscheiden.

PARRY war genau wie ELIZA jedoch nur in einem sehr eingeschränkten Bereich fähig, einen Menschen zu simulieren. PARRY verzichtete auf Desambiguierungsmechanismen oder auf syntaktisches Parsing, da es lediglich nach Schlüsselwörtern sucht. Colby strebte mit PARRY ebenso wenig den Turing Test an, wie Weizenbaum. Es war sein Ziel eine Theorie über typisch paranoides Kommunikationsverhalten durch die Computersimulation zu bestätigen.

2.3.3 Chatbots in den 90-iger Jahren

Die Zeit der Chatbots begann gegen Anfang der 90-iger Jahre. Joseph Weintraub gewann mit seinem Programm **PC Therapist** 1991, 1993 und

1995 die Bronzemedaille bei den Loebner Wettbewerben. Sein Programm war deutlich an ELIZA angelehnt. Der wesentliche Unterschied zu ELIZA beruhte weniger auf der verwendeten Programmiertechnik, als auf Vokabular und Persönlichkeit des Chatbots. Die Einzelheiten über die Funktionsweise sind nicht zugänglich, weil Weintraub sein Programm auch noch heute vermarktet.

Thomas Whalen ging im Jahr 1994 mit seinem Programm **TIPS** als Sieger aus dem Loebner Wettbewerb hervor. Den Anforderungen eines thematisch eingeschränkten Turing Test kam TIPS sehr entgegen, da er sich sein Aufbau sehr stark an der Architektur klassischer Datenbanksysteme orientierte. Jason Hutchens gewann in Jahr 1996 **HeX** die Bronzemedaille. Die Nutzereingabe wird Satz für Satz analysiert und es wird nach Schlüsselwörtern gesucht. Das Programm wird von dem weiteren Modul MegaHal, der eigentliche Chatbot im System, unterstützt. Hutchens modelliert Sprache in Form einer Markov-Kette mit der sich Zeichenketten generieren lassen. Um die Eingabe und die generierten Antworten in Zusammenhang zu bringen, wird diese auf Schlüsselwörter untersucht. Auf Basis dieser Schlüsselwörter sorgen verschiedene Algorithmen für die Generierung einer Antwort. HeX bzw. MegaHal sind im Unterschied zu den vorangegangenen Siegern selbstständig lernfähig.

Auch **FRED bzw. Albert One**, die Sieger von 1998 und 1999 sind lernfähig. FRED wurde von Robby Garner entwickelt und kann als natürlich-sprachige Schnittstelle zu unterschiedlichsten Programmen eingesetzt werden. Sein Wissen bezieht er aus einer großen Datenbank mit Sätzen, auf die er zuvor im Gespräch getroffen ist. Antworten sind auf diese Sätze zugeordnet. Besteht die Eingabe aus einem unbekannten Satz, wird nach einem möglichst ähnlichen Satz in der Datenbank gesucht und dementsprechend die Antwort ausgegeben (vgl. Storp 2002, S.12ff.).

2.3.4 ALICE

ALICE (engl. Artificial Linguistic Internet Computer Entity) wurde von Richard Wallace im Jahr 1995 entwickelt. Es ist ein Open-Source-Chatbot-Programm zur Verarbeitung natürlicher Sprache (vgl. Deshpande et al. 2017, o. S.). ALICE basiert auf XML Wissensbasen bzw. einer XML-Sprache die der HTML-Struktur ähnelt (Deshpande et al. 2017, o. S.; Storp 2002, S.21). Der Vorteil ist, dass AIML leicht und intuitiv erlernbar ist und ist um neue Befehle erweiterbar. Neben dem modularen Aufbau besteht ALICE aus einer ganzen Reihe von Programmen, die in einer Client-Server-Architektur miteinander verbunden sind. ALICE generiert gesprochene Antworten und lernt die Eingaben in gesprochener Sprache zu verstehen (vgl. Storp 2002, S.20f.).

Die Dialoge von ALICE wirken natürlicher, weil sie nicht zur Laufzeit aus einzelnen Wörtern generiert werden, sondern weil die einzelnen Antworten aus den von Menschen geschriebenen Sätzen zusammengesetzt werden. Zu kommerziellen Zwecken wird das ALICE-System erfolgreich eingesetzt. Dies beruht auf seinem einfachen Ansatz der Wissensrepräsentation und den darauffolgenden Algorithmen und auf dem zugrunde liegenden Lizenzmodell. ALICE gehört derzeit zu den fortschrittlichsten Chatbot-Systemen und hat schon mehrere Male bei den Wettbewerben von Loebner als bestes System hervorgetan (vgl. Möbus et al. 2006, S.83).

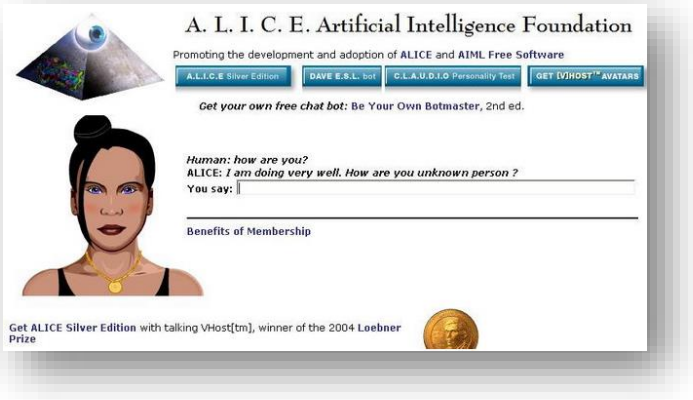

Abbildung 2: ALICE Chatbot (Mobile app design and development company, 2019)

Es wurde in diesem Kapitel kurz ein Überblick über die erfolgreichsten Chatbots und deren EntwicklerInnen gegeben. Im anschließenden Kapitel 2.4 wird nun ein Einblick in die Funktionsweisen von Chatbots gegeben. Thematisiert werden unter anderem auch die Wissensdatenbanken, die das Kernstück eines Chatbots ausmachen.

2.4 Funktionsweise von Chatbots

Die heutigen Chatbots sind modular aufgebaute Reiz-Reaktions-Systeme, die mit einer internen Musterdatenbank den sprachlichen Input abgleichen und im Anschluss entsprechende Antworten ausgeben. Die so genannte Wissensdatenbank ist auch heute das Herzstück des Chatbots, in dem Erkennungsmuster, Stichwörter und Antworten gespeichert sind. Der Gesprächsablauf wird vom eigentlichen Programm gesteuert und koordiniert die Ein- und Ausgabe, die Aktivierung der Wissensdatenbank und gegebenenfalls weitere Modelle wie die Ausgabe gesprochener Sprache. Außerdem stellen vor allem kommerzielle Systeme einen Editor zur Verfügung, der dem Anwender den Aufbau und die Pflege der Wissensdatenbank erleichtert. In der Protokollfunktion, über die jeder Chatbot verfügt, sind sämtliche Dialoge gespeichert und diese steht den EntwicklerInnen zur Auswertung bereit. Die einzelnen Systeme unterschieden sich allerdings in Größe und Flexibilität ihrer „Wissensdatenbanken" sowie in der der Leistungsfähigkeit ihrer Steuerungsprogramme.

Während einige Systeme die Antworten aus Rümpfen generieren, wählen hingegen andere die Antworten per Zufallsprinzip aus. Manche Chatbots vergleichen mit der Musterdatenbank nur einen Teil einer Eingabe, andere Chatbots analysieren alle Teile der Eingabe, die mit den Antworten kombiniert werden. Die meisten Chatbots führen Protokoll über ihre Gespräche und schlagen darauf beruhende neue Stichwörter oder Erkennungsmuster vor. Die Entscheidung wird aber vom Menschen getroffen, ob diese in die Wissensdatenbank aufgenommen werden oder nicht. Dieses Verfahren wird

als überwachtes Lernen bezeichnet. Einige Systeme erweitern selbstständig ihre Datenbank anhand der Protokolle. Wenn für die Eingabe kein Erkennungsmuster vorliegt, können einige Chatbots auch externe Datenbanken nutzen (vgl. Storp 2002, S.19f.).

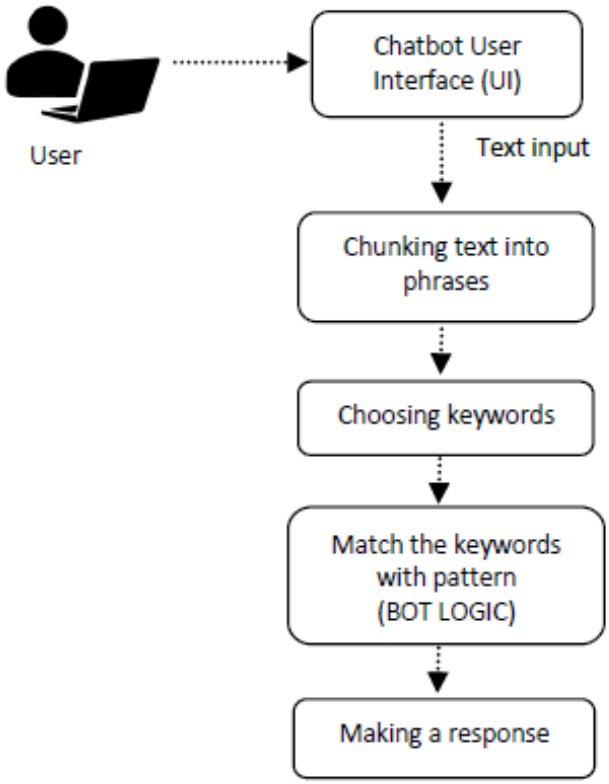

Abbildung 3: Chatbot-Systemprozesse (Ahmad et al., 2018 S.9)

Der Erfolg eines Chatbot-Systems hängt also hauptsächlich von der Qualität des Wissens und von der Wissensmenge in der Wissensbasis ab. Die Erstellung der Wissensbasis ist eine aufwändige Arbeit. Durch eine Verbesserung der Algorithmen des Systems (z.B. verbesserte Analyse der Eingaben) kann aber diese aufwändige Arbeit zeitlich verkürzt werden (vgl. Möbus et al. 2006, S.83). Durch welche Maßnahmen aber versucht wird, den

Chatbot menschenähnlich erscheinen zu lassen, wird im nachfolgenden Kapitel 2.5 thematisiert.

2.5 Menschliche Züge

Um Chatbots menschlicher zu gestalten oder menschlicher wirken zu lassen, gibt es eine Vielzahl von klugen Maßnahmen, wie beispielsweise eine verlangsamte Antwortzeit. Obwohl die meisten Chatbots aufgrund leistungsstarker Computer sehr schnell arbeiten, können sie auch eine schnelle Antwort bzw. Informationen liefern. Weil Chatbots aber ein Gespräch mit einem Menschen vortäuschen, der für seine Eingabe mittels Tastatur Zeit braucht, kann die Ausgabe in verzögert erfolgen. Oft werden auch Tippfehler in Chatbots eingebaut oder sie sind in der Lage, kleine Geschichten zu erzählen und Gesprächsthemen einzubringen. Viele Chatbots werden auch mit fremdsprachlichen Floskeln oder mit Zitaten versorgt. Nicht zuletzt scheint es erfolgsversprechend, den Chatbot eine virtuelle Identität als Persönlichkeit zu verschaffen (vgl. Storp 2002, S.26ff.).

2.6 Klassifizierungsmerkmale

Anschließend werden einige Klassifizierungsmerkmale von Chatbots tabellarisch dargestellt und kurz erläutert, was darunter verstanden wird.

Regelbasiert	Selbstlernend
Um Antworten zu geben und um die NutzerInnen zu verstehen, greift der Chatbot auf ein bestehendes Set aus Texten und Antworten	Durch „Machine Learning" im Zuge von vielen Labor-Trainings und Konversationen mit Menschen lernt der Chatbot selbstständig neue

zurück, daher sind sie nicht künstlich intelligent.	Ideen und Verknüpfungen, solche sind künstlich intelligent.

Tabelle 1: regelbasiert und selbstlernend

Universell	Themenspezifisch
Es gibt keine inhaltliche Eingrenzung der Konversation. Theoretisch muss der Chatbot von der Taxibuchung bis hin zum Wetter alles verstehen und souverän meistern, was die Programmierung aufwändiger macht.	Die Erwartung an den Chatbot und das Thema für die Unterhaltung ist begrenzt.

Tabelle 2: universell und themenspezifisch

Eigenständig	In Messenger integriert
Als Apps oder auf Webseiten werden eigenständige Chatbots abgerufen.	Sodass die NutzerInnen auf sie bequem und intuitiv zugreifen können, werden Chatbots vermehrt in Messenger-Plattformen, wie beispielsweise Skype und Telegramm, integriert.

Tabelle 3: eigenständig und in Messenger integriert

Akustisch	Schriftlich

Die NutzerInnen können mit dem Chatbot sprechen (z.B. Alexa oder Siri).	Mit dem Chatbot wird schriftlich kommuniziert (vgl. Futuregram 2016, o. S.)

Tabelle 4: akustisch und schriftlich

2.7 Einsatzgebiete

In diesem Abschnitt wird beschrieben, in welchen Gebieten Chatbots bereits erfolgreich zum Einsatz kommen.

Chatbots werden bereits in vielen Bereichen flexibel eingesetzt. Chatbots werden im Bereich der Bildung, im Gesundheitswesen und in den verschiedensten Unternehmensbranchen, insbesondere für Marketingzwecke, eingesetzt. Viele Firmen haben Chatbots in ihre Systemumgebung eingebettet, wie beispielsweise Facebook seinen Facebook Messenger, Google seinen Google Assistenten, Microsoft seine Software Cortana oder wie zum Bespiel der Konzern Apple seine Software Siri.

Beispielsweise kann ein Chatbot im Bildungsbereich als ein intelligenter Tutor für den Online-Lernenden als Bildungswerkzeug eingesetzt werden. Ebenso können Chatbots eingesetzt werden, um Probleme zu lösen oder eine große Anzahl von Schüler und Schülerinnen individuell zu unterstützen. Im Gesundheitswesen werden Chatbots unterstützend eingesetzt, um etwa die langfristige Einhaltung von Gesundheitsförderungsmaßnahmen zu erleichtern. Ein Chatbot arbeitet beispielsweise als Gesprächsassistent und fungiert als bidirektionaler Kanal zwischen den GesundheitsexpertInnen und den BenutzerInnen, wenn die BenutzerInnen durch die Beratung zu gesunder Ernährungsgewohnheiten, körperlichen Aktivitäten, Lebensmittelzubereitung und dem Kauf beraten wird (vgl. Ahmad et al. 2018, S.8).

Auch in der Banken-, Finanz- und Versicherungsindustrie werden Chatbots erfolgreich eingesetzt. Im Rahmen des Internet-Bankings kann über ein Chat-Interface auf normale Bankprozesse zugegriffen werden, einschließlich Aktivitäten wie das Finden einer Filiale, das Überprüfen des Guthabens oder einer Geldüberweisung auf ein anderes Konto. Kundensupport-Anwendungsfälle wie das Sperren einer gestohlenen Kreditkarte oder das Anfordern einer neuen Karte oder können einfach über einen Chatbot durchgeführt werden. Versicherungstätigkeiten zwischen dem Kunden und der Versicherungsgesellschaft beinhalten viel Kommunikation. Die Daten, die zwischen den Parteien ausgetauscht werden, können automatisiert werden, da sie strukturiert sind. Einige der Anwendungsfälle sind z. B. das Erhalten von Informationen über andere Versicherungsprodukte oder die Registrierung eines Versicherungsanspruchs.

Auch in der Reiseindustrie findet viel Kundeninteraktion vor dem Verkauf statt. Der Preis ist einer der Haupttreiber des Umsatzes im Reisebereich. Käufer sind stets bestrebt, Preise zu optimieren, die sie bei der Buchung eines Fluges oder Hotels zahlen. Ein Anwendungsfall wäre, einen Chatbot zu entwickeln, der mit Backends spricht, um Hotel- und Flugpreise zu erhalten und der alle Preise im Auge behält. Eine Benachrichtigung kann ausgelöst werden, sobald die Preise für einen bestimmten Sitzplatz steigen oder fallen. Der Vorteil des Chats ist, dass alle Änderungen an ihnen leicht nachvollzogen werden können, da der gesamte Kontext der vorherigen Suchen auf dem Bildschirm sichtbar ist.

Es gibt viele Anwendungsfälle in der Gastronomie, die auf einem Chatbot automatisiert werden können. Die meisten Anfragen beziehen sich in der Gastronomie auf Tischreservierungen. Die meisten Tischreservierungen werden auch heute noch über ein Telefon abgewickelt. Für dieses Problem scheint ein Chatbot gut geeignet zu sein. Es ist für die Kunden und Kundinnen sicher bequem auf einen Chatbot zuzugreifen, um eine Tischreservierung vorzunehmen.

In den Anwendungsfällen für E-Commerce gibt es vor allem zwei Funktionen, die ein Chatbot ausführen kann: Kundensupport und Produktsuche. Für E-Commerce ist die Automatisierung des Kundensupports ein riesiger Markt und mit den Fortschritten im Sprachverständnis von Computern können bald die gesamten Kundensupportanfragen von automatisierten Systemen bearbeitet werden.

Chatbots können auch beim Bezahlen von Rechnungen unterstützend helfen und bieten viel Wert für den Endkunden. Zum Beispiel können Telekommunikations- und Elektrizitätsunternehmen davon profitieren, indem sie einen Chatbot für ihre Kunden und Kundinnen auf verschiedenen Plattformen einführen und grundlegende Dienste zum Abrufen von Rechnungen zusammen mit der Integration einer Zahlungslösung anbieten (vgl. Schlündt 2018, o. S.).

Auch die Modebranche setzt bereits auf Chatbots die ihre Kunden und Kundinnen als Styling Assistent beim Kauf und beim Finden der passenden Outfits unterstützen. Für die verschiedensten Anlässen finden die Programme die richtigen Kleidungsstücke und eine problemlose Weiterleitung zum Online-Shop ist garantiert. Transportunternehmen setzten Chatbots auch bereits ein, damit Kunden und Kundinnen bequem ihre Taxifahrt bestellen können. Auch im Entertainmentbereich gibt es zahlreiche Chatbots die bereits eingesetzt werden, wie beispielsweise zur Promotion von Kinofilmen, zur Informationsbereitstellung zu Kinostandorten, aktuelle Trailer und Filme und vieles mehr (vgl. Onlim GmbH 2018, o. S.).

Zusammenfassend ist anzumerken, dass Chatbots bereits in verschiedenen Bereichen eingesetzt werden und aus meiner Ansicht als Entwickler steckt sehr viel Potential in ihnen. Zukünftig werden Chatbots in weiteren Bereiche eingesetzt werden und können uns auch Aufgaben abnehmen, von denen wir zurzeit noch keine Vorstellung haben.

3 IDERBLOG CHATBOT

Dieses Kapitel widmet sich dem Prototypen des IDeRBlog-Chatbots. Zu Beginn wird im Abschnitt 3.1 die Aufgabenstellung dieser Masterarbeit kurz aufgezeigt. Anschließend werden im Abschnitt 3.2 die eingesetzten Technologien des IDeRBlog-Chatbots und im Abschnitt 3.3 wird die eingesetzte Funktionen und Bibliotheken beschrieben. Jeweils werden nach den Abschnitten 3.2 und 3.3 die Entscheidungen kurz erläutert, warum diese Wahl der Technologie getroffen wurde.

3.1 Definition der Aufgabe

Die Aufgabenstellung im Rahmen dieser Masterarbeit umfasst die Entwicklung und die Programmierung eines Chatbots-Prototypen, der in der Lern-App IDeRBlog integriert wird.

IDeRBlog (siehe Abbildung 4) gehört zu den Lern-Apps des Learning LAB des TU Graz Instituts für Lehr- und Lerntechnologien (weitere Informationen unter: https://learninglab.tugraz.at), welches verschiedene Lerntools im Bereich Mathematik, Deutsch, Englisch, Informatik und Zeichnen für Lernende und Lehrende zur Verfügung stellt. SchülerInnen und LehrerInnen können die Lern-Apps zum Üben kostenlos benutzten.

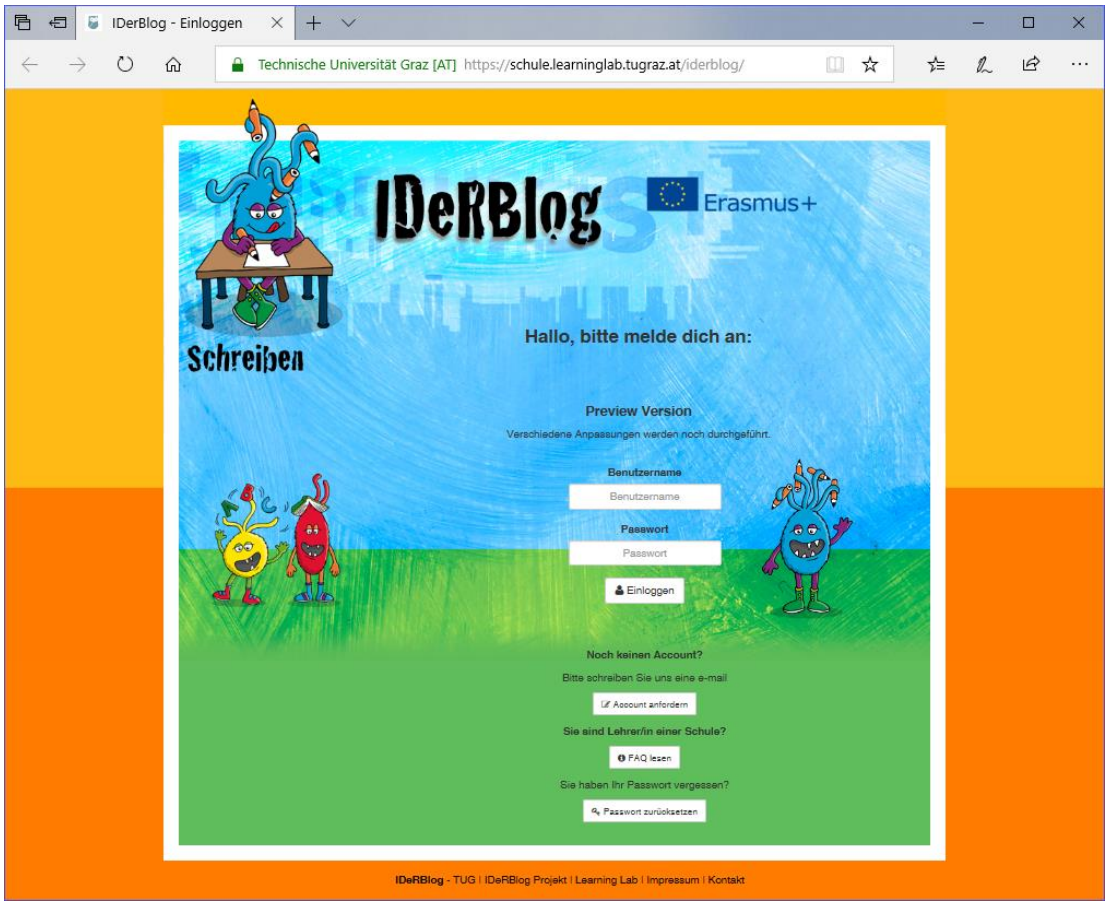

Abbildung 4: IDeRBlog

Der IDeRBlog-Chatbot der für den IDeRBlog entwickelt wird, sollte auf Rechtschreibfehler, die die VerfasserInnen machen, aufmerksam machen und Verbesserungsvorschläge bieten. Die SchülerInnen sollen durch die Fehleranalyse mit dem IDeRBlog-Chatbot durch Wörtereingabe interagieren und auch durch das Stellen von Fragen zusammen mit dem IDeRBlog-Chatbot korrekte Wörter verfassen. Die Interaktion, Eingabe und die Bedienung des IDeRBlog-Chatbot soll für die BenutzerInnen sehr einfach gestaltet sein. Die SchülerInnen sollte auch das Gefühl gegeben werden, wie mit einem Betreuer oder Tutor kommunizieren zu können, der sie bei ihren Lernprozessen bestmöglich unterstützt.

3.2 Eingesetzte Technologien und IDE

Dieser Abschnitt widmet sich der Hyper Text Language (Abschnitt 3.2.1), die für die Struktur des IDeRBlog-Chatbot-Webfrontend verwendet wurde. Das Layout wurde mit CSS (Cascading Stylesheets) definiert, welches im Abschnitt 3.2.2 beschrieben wurde. Anschließend wird im Abschnitt 3.2.3 die Programmiersprache JavaScript beschrieben, weil sie für die Kommunikation zwischen Webfrontend und Webbackend für den Nachrichtenaustausch, durch Verwendung der Websockets, eingesetzt wurde. Im Abschnitt 3.2.4 wird das Bootstrap-Framework erläutert, welches für die Entwicklung des responsiven Webfrontends für den IDeRBlog-Chatbot zusätzlich verwendet wurde. Im Abschnitt 3.2.5 wird Java und das Java Spring Boot – Framework thematisiert, weil es für die Entwicklung des Webbackends eingesetzt wurde.

3.2.1 Hyper Text Markup Language (HTML)

Die zentrale Auszeichnungssprache des World Wide Web ist HTML. HTML war ursprünglich und hauptsächlich als Sprache für die semantische Beschreibung wissenschaftlicher Dokumente gedacht (vgl. W3C 2019, o.S.). Heutzutage wird mit HTML jede Webseite erstellt. Die Struktur einer Webseite und die Bedeutung (die Semantik) einzelner Komponenten auf einer Webseite (vgl. Abbildung 5) werden mit Hilfe von HTML über die HTML Elemente festgelegt.

Zum Beispiel beschreiben diese Komponenten welcher Bereich auf der Webseite den Hauptinhalt und welcher Bereich die Navigation darstellt und definieren Komponenten wie Tabellen, Eingabefenster, Formulare, Listen und Schaltflächen (vgl. Ackermann 2018, S.55).

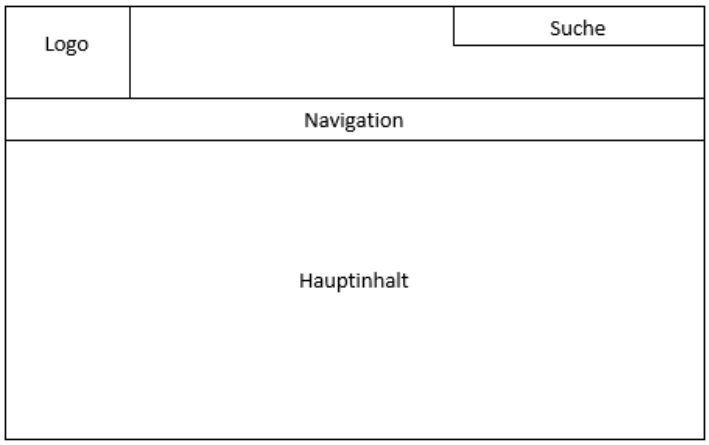

Abbildung 5: Beispiel einer Struktur einer Webseite

HTML hat zahlreiche Elemente. Wenn diese Elemente mit spitzen Klammern umschlossen werden, nennt man sie HTML-Tags. Ein HTML-Element ist zum Beispiel wie folgt definiert: <title>Meine HTML-Seite</ title>, wobei es am Anfang mit einem Start-Tag (hier: <title>) beginnt und am Ende mit einem Ende-Tag (hier: </title>) abschließt. Dazwischen wird der Inhalt „Meine HTML-Seite" geschrieben. Eine Webseite besteht aus mehreren HTML-Dokumenten, die untereinander verlinkt werden. Dies ist gleichzeitig die Stärke der HTML, weil sie die Navigation auf der Webseite und im Web ermöglicht. HTML-Dokumente können mit ein beliebigen Texteditor geschrieben werden. Der Quellcode eines HTML-Dokuments ist zwar lesbar (siehe Auflistung 1) aber zum Betrachten empfiehlt sich ein Webbrowser (Abbildung 6).

```
01 <!DOCTYPE html>
02 <html lang="de">
03 <head>
04 <meta http-equiv="Content-Type" content="text/html; charset=utf-
8"/>
05 <title>Meine HTML-Seite</title>
06
07 </head>
08 <body>
09 <h1>Das ist meine Überschrift</h1>
10 <p>Das ist mein Absatz</p>
11 <a href="https://www.w3schools.com/">Das ist eine Verlinkung</p>
12 </body>
13 </html>
```

Auflistung 1: einfaches HTML-Dokument mit seinen Grundelementen

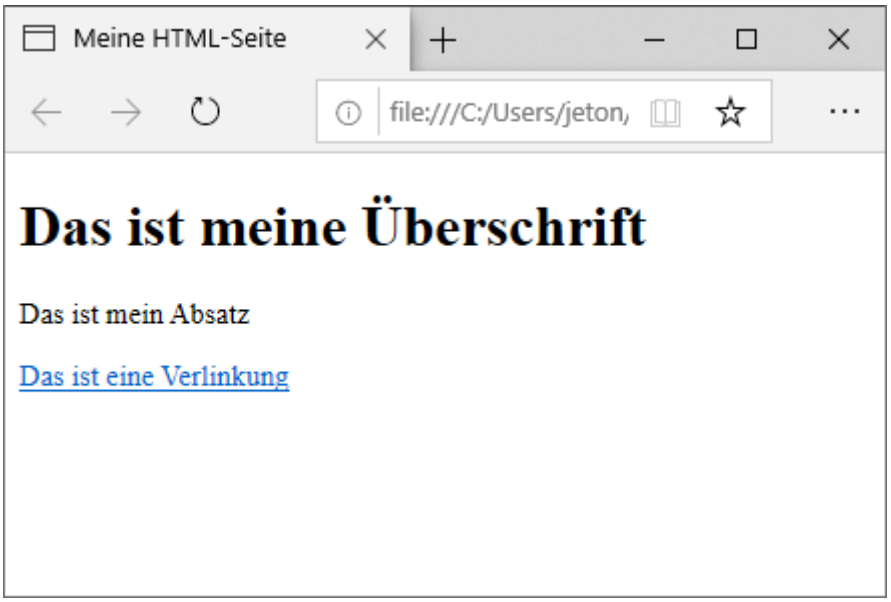

Abbildung 6: Browseransicht des HTML-Dokuments

Wie in der Abbildung 6 ersichtlich, ist die Browseransicht eines HTML-Dokuments, welches nur mit HTML-Elementen umgesetzt wurde, nicht sehr ansprechend und bedienungsfreundlich für die BenutzerInnen. Es fehlt an Formatierung, Schriftarten, Farben und Positionierungen der HTML-Elemente. Um dieses zu erreichen werden Cascading Stylesheets (CSS) verwendet, die im nächsten Abschnitt 3.2.2 kurz erörtert werden.

HTML wurde im Zuge dieser Masterarbeit eingesetzt, um die Struktur des Webfrontends und die dazugehörigen Eingabe- und Ausgabemasken für den IDeRBlog-Chatbots umzusetzen. Weitere Details der Umsetzung werden im Kapitel 4.2 ausführlich beschrieben und mit Grafiken dargestellt.

3.2.2 Cascading Stylesheets (CSS)

Unter Cascading Stylesheets werden Formatvorlagen verstanden, die das Layout von Webseiten definieren. Sie ergeben zusammen mit dem HTML-Markup eines Dokuments das Aussehen einer Webseite. Mit CSS werden grundlegende Konzepte von HTML bewahrt und gleichzeitig wird mehr Flexibilität im Design erreicht. Eine wichtige Voraussetzung ist dabei die

Trennung von Struktur und Gestaltung der Webseite. Durch CSS werden die Formatierung, Platzierung, Größe bzw. weitere gestalterische Optionen von Elementen eines HTML-Dokuments möglich. Um zu zeigen, wie mächtig CSS ist, wird in der folgenden Abbildung 7 einen Minion dargestellt (abrufbar unter https://codepen.io/rachel_web/pen/pjzowB), der rein nur mit CSS und HTML erstellt worden ist (vgl. Laborenz 2015, S.17ff.).

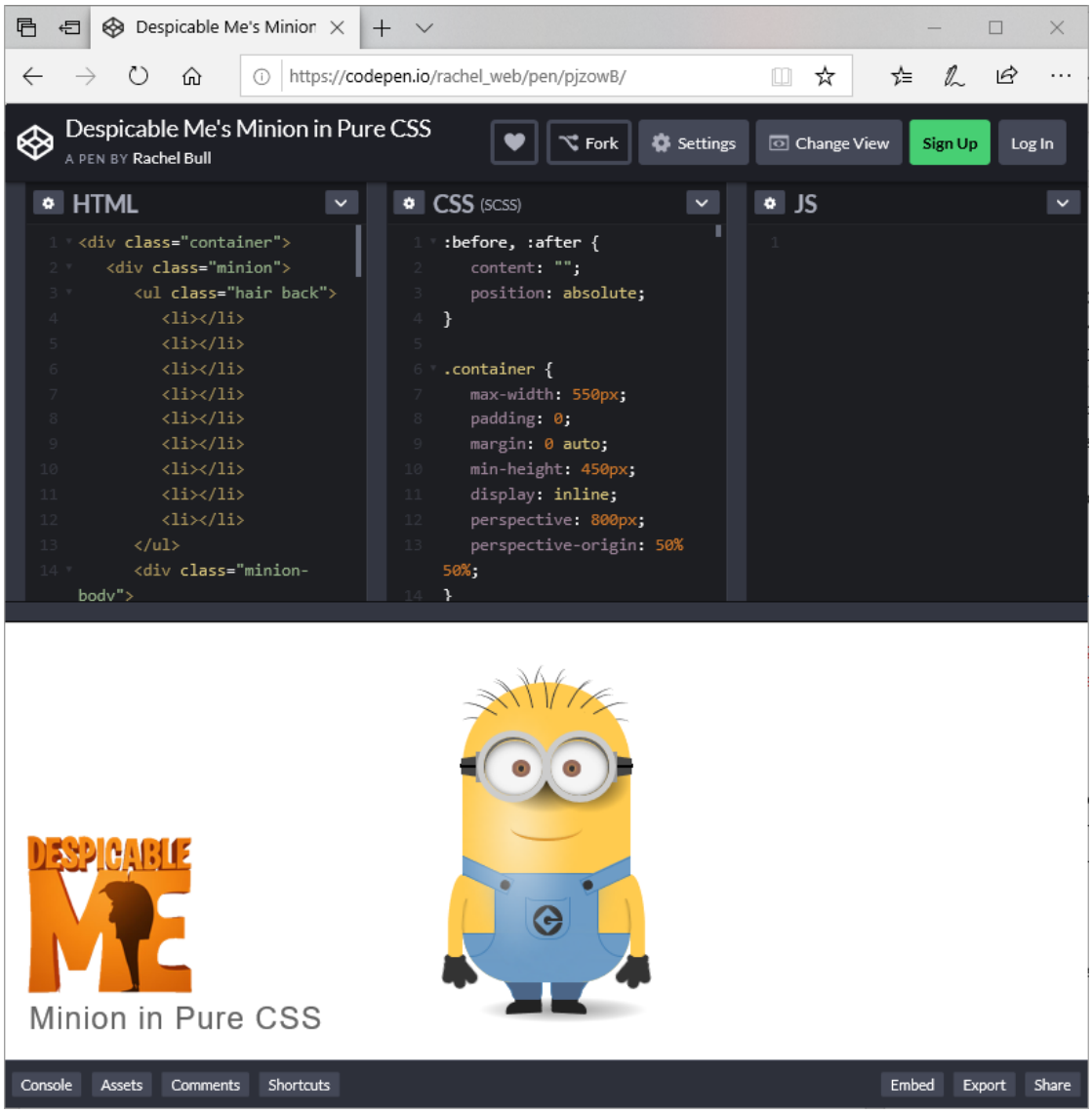

Abbildung 7: Minion erstellt mit HTML und CSS (in Anlehnung an Laborenz 2015, S.19)

„Eine CSS-Anweisung besteht immer aus zwei Teilen – dem Selektor und der Deklaration (Laborenz 2015, S.23).“ Der Selektor p gefolgt von zwei geschweiften Klammern {} gibt dem Browser die Anweisung welches HTML-Element ausgewählt werden soll, während die Deklaration vorgibt, wie dieses ausgewählte Element formatiert werden soll (z.B. Größe und Farbe des Elements) und ist durch geschweiften Klammern {} umschlossen. In der nachfolgenden Auflistung 2 wird der Aufbau einer CSS-Anweisung dargestellt (vgl. Laborenz 2015, S.23).

```
p {color: red;font-size: 20px;}
```

Auflistung 2: Aufbau einer CSS-Anweisung

Mit dem aktuellen CSS3 ist es sogar möglich, HTML-Elemente zu animieren (vgl. Laborenz 2015, S.18), was bis jetzt nur mit der Programmiersprache JavaScript möglich war. Da JavaScript auch in der Erstellung des Prototypen-Chatbots die Server-Client-Kommunikation ermöglicht, wird im nächsten Abschnitt 3.2.3 eine kurze Einführung in die Programmiersprache JavaScript gegeben.

CSS wurde in der Entwicklung des Chatbots im Web-Frontend eingesetzt, um die Eingabe- und Ausgabemasken zu gestalten. Berücksichtigt wurde die Anpassung der Schriftfarben an das bereits bestehende Layout des IDeRBlogs, die passende Schriftgröße und die richtige Positionierung der Masken.

3.2.3 Java Script und Websocket

Die Programmiersprache des Web ist JavaScript und wird insbesondere bei modernen Webseiten verwendet. Alle modernen Browser enthalten

außerdem einen JavaScript-Interpreter. JavaScript ist damit die am weitesten verbreitete Programmiersprache der Geschichte (vgl. Flanagan 2012, S.VII).

> „JavaScript gehört zu den drei Technologien, die alle Web-Entwickler lernen müssen: HTML, um den Inhalt von Webseiten zu definieren, CSS, um festzulegen, wie die Inhalte dargestellt werden sollen, und JavaScript, um das Verhalten der Elemente einer Webseite zu steuern. Mit der Entwicklung von Node (…) wird JavaScript außerdem auf Webservern immer wichtiger (Flanagan 2012, S.VII)."

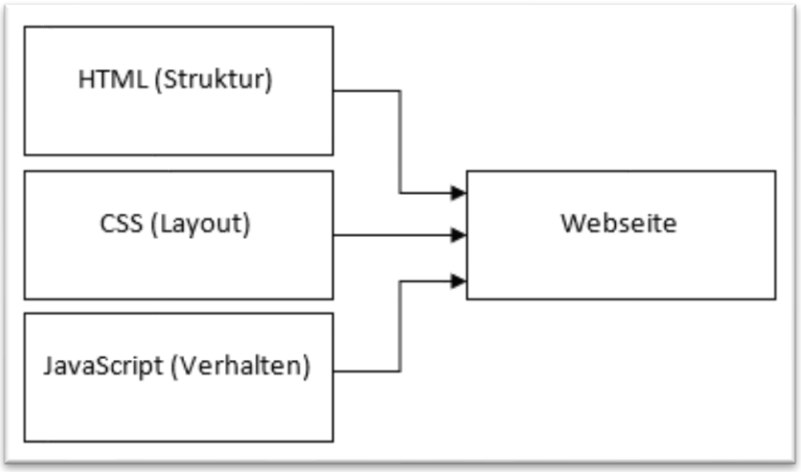

Abbildung 8: Kombination von HTML, CSS und JavaScript auf einer Webseite (Quelle: in Anlehnung an Ackermann 2018, S.57)

Bevor die Kommunikation mit Websockets in JavaScript thematisiert wird, wird kurz in die HTTP-Kommunikation eingeführt.

HTTP-Kommunikation

Bisher war die HTTP-Kommunikation zwischen Webserver und Webbrowser auf das Anfrage-Antwort-Modell beschränkt: eine Anfrage wurde vom Browser gestellt und der Webserver sendete eine Antwort. Eine aktive Sendung von Daten vom Webserver an den Browser ist nicht möglich. Lediglich durch wiederholtes Anfragen (Polling) beim Server kann der Browser nachfragen, ob es neue Daten gibt (vgl. Schwichtenberg Holger 2019, o.S.). Eine vernünftige Integration von Echtzeit bei den traditionellen

Anwendungen fehlt allerdings. Häufig wird In Webanwendungen durch Polling oder Long Polling die Echtzeit simuliert. Diese Techniken (oder Hacks) werden oft durch die Begriffe Comet oder Bayeux beschrieben. In einem festgelegten Intervall, beispielsweise alle zwei Sekunden, wird der Server im Fall von Polling (siehe Abbildung 9) gefragt, ob er neue Informationen hat. Beim Long Polling (siehe Abbildung 10) wird zum Server eine separate HTTP-Verbindung erst dann geschlossen, wenn neue Daten verfügbar sind (vgl. Weßendorf 2011, o.S.).

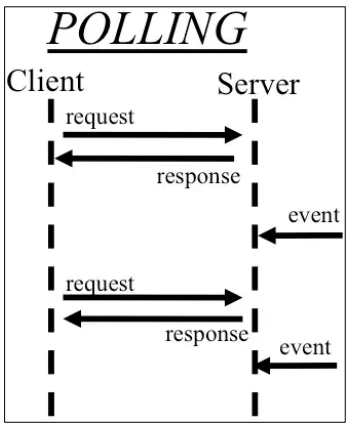

Abbildung 9: Ablauf von Polling (Weßendorf, 2011, o.S.)

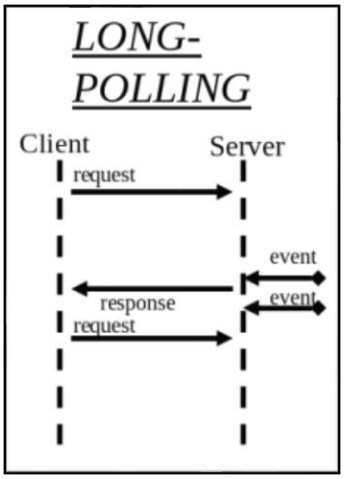

Abbildung 10: Ablauf von Long Polling (Weßendorf, 2011, o.S.)

Nachdem der Browser die Daten verarbeitet hat, wird ein neuer Request zum Server gesendet, um auf weitere Updates zu warten. Von Natur aus ist HTTP

"nur" halbduplex. Das bedeutet, dass für die bidirektionale Kommunikation zwischen Server und Browser ein separater HTTP Request für jede Richtung benötigt wird. Natürlich wird dadurch eine Menge Overhead erzeugt. Auf dem Server kommt Neben dem zusätzlichen I/O-Handling noch der Netzverkehr hinzu: HTTP Request/Response Header können schnell ein paar Hundert Bytes veranschlagen. Die eigentlich wertlose Information, dass es keine Änderungen am Zustand des Servers gab, kommt ab und an noch zusätzlich hinzu. die effiziente Programmierung von Webanwendungen, die auf einen schnellen Datentransfer angewiesen sind, werden durch diese Tatsachen erschwert (vgl. Weßendorf 2011, o.S.).

Kommunikation mit Websockets

Zum Versand von Nachrichten von Webservern und Webbrowser wird Websockets, ein relativ junger, neuer W3C (World Wide Web Consortium) Standard für die bidirektionale Kommunikation zwischen Webserver und Webbrowser eingesetzt (vgl. Schwichtenberg Holger 2019, o.S.). WebSocket ist ein (voll-)duplexer und bidirektionaler Kommunikations-standard und besteht aus einer Client-API und einem Netzwerkprotokoll. Der große Vorteil von WebSocket steckt in dem Wort vollduplex. Die bidirektionale Kommunikation zwischen Client und Server läuft über genau einen Kommunikationskanal ab. Sobald die WebSocket-Verbindung aufgebaut ist, können Client und Server gleichzeitig miteinander kommunizieren. Der Verbindungsaufbau zwischen Server und Client erfolgt mit dem WebSocket Protokoll Handshake (vgl. Weßendorf 2011, o.S.).

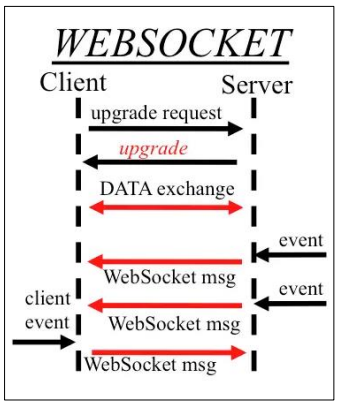

Abbildung 11: Der Websocket Handshake (Weßendorf, 2011, o.S.)

Mit JavaScript muss die Interaktion clientseitig eingeleitet werden und die Nachrichten des Webserver behandelt werden. Dafür benötigt man das Objekt WebSocket mit den Operationen OnOpen(), OnClose() und OnMessage() (vgl. Schwichtenberg 2019, o.S.).

Der Aufruf mit der Operation OnOpen() erfolgt nach Verbindungsherstellung, der Aufruf erfolgt sobald die Verbindung vom Server oder vom Client geschlossen wurde mit der Operation OnClose() und mit OnMessage() erfolgt der Aufruf, sobald Nachrichten vom Server im Client angekommen sind. Mit der Operation OnError() erfolgt der Aufruf im Fehlerfall.

Der Client kann nach dem erfolgreichem Aufbau der WebSocket-Verbindung über den bidirektionalen Kommunikationskanal Nachrichten mit der JavaScript-Funktion send() an den Server senden. Die OnClose()-Methode kann aufgerufen werden, sobald der Client die Verbindung beenden möchte. Im Vergleich zu HTTP ist der Overhead von WebSocket gering. Statt mehrerer hundert beträgt er pro Nachricht genau zwei Byte (vgl. Weßendorf 2011, o.S.).

Java wurde für die Programmierung des Webfrontends eingesetzt. Die Kernaufgabe ist das Senden und Empfangens der Nachrichten durch Verwendung von Websockets. Zusätzlich wurden die Interkationen und das Verhalten der Ein- und Ausgabemasken mit JavaScript realisiert.

3.2.4 Bootstrap

„Bootstrap ist ein Frontend-Framework“ (Weigel 2019, o.S), mit den Webseiten gestaltet werden. Es werden CSS- und HTML-Vorlagen bereitgestellt, um unterschiedlichste Webseiten-Elemente darzustellen. Dazu gehören ein Grid-System für Layouts sowie Formulare, Buttons, Tabellen und eine Navigation. Durch JavaScript-Module ist es möglich, Interaktionen (z. B. eine Bilder-Slideshow, Tabs und Dialogboxen) in die Website einzubinden. Bootstrap bietet zudem alle Voraussetzungen, um responsive Webdesigns zu gestalten, um eine optimale Darstellung auf

Tablets oder Smartphones zu gewährleisten. Auf der Webseite https://getbootstrap.com (abgerufen am 30.03.2019) können nähere Informationen über Bootstrap eingeholt werden bzw. steht das Programm dort zum Download (inzwischen liegt die aktuelle Version v4.3.1 vor) bereit.

Zur Geschichte von Bootstrap kann gesagt werden, dass das Programm im Jahr 2010 unter dem Namen „Twitter Bootstrap" von Twitter entwickelt wurde, mit dem Ziel, eine einheitliche Bibliothek für die Gestaltung von Weboberflächen zu schaffen. Twitter entschloss sich 2011 dazu, das Bootstrap Framework als Open Source Projekt zu veröffentlichen.

Das Bootstrap Framework besteht im einfachsten Fall aus einer JavaScript-Datei (bootrsap.min.js) und einer CSS-Datei (bootrap.min.css). In Bootstrap können viele Design-Elemente nur mit CSS-Regeln umgesetzt werden. JavaScript wird beispielsweise für Dialogboxen und andere Komponenten benötigt (vgl. Weigel 2019, o.S.). „Bootstrap besteht aus einzelnen Komponenten, (…) ist modular aufgebaut und wird in folgende vier Bereiche gegliedert:

- Layout
- Inhalt
- Komponenten
- Werkzeuge." (Weigel 2019, o.S.).

Zur kurzen Einführung in Bootstrap wird nun kurz das Raster beschrieben, da das Raster das Kernstück von Bootstrap ist und eine optimale Anpassung an den Bildschirm des Endgerätes ermöglicht. Es besteht aus Spalten (.col) und Raster (.row), wobei die Spalten mit Hilfe von CSS-Klassen passend für das jeweilige Endgerät angeordnet werden können. In der folgenden Abbildung 12: Bootstrap-Raster (Weigel 2019, o.S.) ist das Bootstrap-Raster mit den verschiedenen CSS-Präfix-Klassen dargestellt. Werden die Spaltenbreiten (XX-Angabe) weggelassen, wird eine Darstellung von Layouts mit unterschiedlich breiten Spalten möglich (Weigel 2019, o.S.).

Displaygröße	Sehr klein	Klein	Mittel	Groß	Sehr groß
	Smartphone (Hochformat)	Smartphone (Querformat)	Tablet	Desktop-PC	Desktop-PC
CSS-Präfix	.col-XX	.col-sm-XX	.col-md-XX	.col-lg-XX	.col-xl-XX

Abbildung 12: Bootstrap-Raster (Weigel 2019, o.S.)

Eine Vermischung von Klassen ist möglich. Beispielsweise kann eine Sidebar, die auf dem Desktop-PC nur 20% der Breite einnimmt, auf diese Weise auf dem Smartphone die Hälfte des Displays abdecken.

Um das Webfrontend passend darzustellen, wurde Bootstrap in der Umsetzung des Prototypens im Webfrontend verwendet. Damit wurde sichergestellt, dass sich die Fenster auf jeden beliebigen Bildschirm optimal darstellen lassen.

3.2.5 Java Spring Boot

Das Spring-Framework bzw. die Spring-Plattform dient dazu, die Entwicklung (Programmierung) mit Java zu vereinfachen, die es ermöglicht, produktionsreife und eigenständige Anwendungen auf Basis des Spring-Frameworks zu bauen (vgl. Simons 2018, o.S.). Spring basiert auf folgende Grundsätze, um dem Programmierer Komfort zu bieten:

Dependency Injection (DI): Die Dependency Injection ist eine (vgl. 1&1 IONOS SE 2019, o.S.) „externe Instanz, die die Abhängigkeiten von Objekten im Vorhinein regelt (…) und bedient sich zu diesem Zweck der Java-Komponenten JavaBeans. Diese fungieren unter anderem als Container zur Datenübertragung, weshalb sie im Spring-Framework als Vorlage für alle verwalteten Ressourcen („beans“) dienen (vgl. 1&1 IONOS SE 2019, o.S.)“

Spring fungiert auf diese Weise als Container, der dem jeweiligen Java-Projekt vorkonfigurierte Klasen inklusive ihrer Abhängigkeiten zur Verfügung stellt.

Aspektorientierte Programmierung (AOP): Spring bietet optional auch einen aspektorientierten Programmierungsansatz, um die Modularität objektorientierter Applikationen zusätzlich zu erhöhen. Komponentenübergreifende Zusammenhänge lassen sich somit syntaktisch strukturieren, die in komplexen Systemen unvermeidbar sind. Das hat zum Vorteil, dass der eigentliche Programmcode von technischen Abläufen wie beispielsweise Validierung, Fehlerbehandlung oder Sicherheit getrennt wird.

Templates: Klassen, für verschiedene Programmschnittstellen, werden in Spring als Templates bezeichnet, die die Arbeit mit API`s vereinfachen, indem sie automatisches Ressourcenmanagement, einheitliche Fehlerbehandlung und andere Hilfestellungen bieten (vgl. 1&1 IONOS SE 2019, o.S.) Folgende Eigenschaften weisen Spring-Frameworks auf (vgl. Simons 2018, o.S.):

- „eigenständige Anwendungen, die keine externen Laufzeitabhängigkeiten mehr haben
- Eingebettete Container (...), so dass keine War-Dateien verteilt werden müssen
- Automatische Konfiguration soweit möglich
- Bereitstellung von nicht funktionalen Eigenschaften, die zur Produktion in der Regel benötigt werden (…)
- Keinerlei Generierung von Code oder Konfiguration.“ (Simons 2018, o.S.).

Cloud-native Anwendungen, die im Hintergrund speziell mit dem Cloud-Modell entworfen wurden, weisen folgende Kernaspekte auf:

- Betrieb und Entwicklung arbeiten Hand in Hand zusammen, sodass Builds, Tests und Releases zuverlässig und oft durchgeführt werden,
- Continuous Delivery, um einzelne Aspekte veröffentlichen und freigeben zu können, wenn sie fertig sind,

- Microservices als architektonischer Ansatz, um Anwendungen als Sammlung kleiner Dienste zu strukturieren, die unabhängig voneinander verteilt, skaliert und aktualisiert werden können, und
- Container als Laufzeitumgebung, die ähnlich wie ein Microservice gestartet, gestoppt und skaliert werden können.

Die 12-Faktor-App ist eine Methode, die wiederkehrende Probleme mit Konfiguration, Portierbarkeit, Skalierung und Verteilung von Anwendungen verhindert soll und ist ein wesentlicher Bestandteil von Cloud-native-Anwendungen.

Für das Webbackend wurde das Java Spring-Framework eingesetzt, weil das IDeRBlog-Framework in dem Spring-Framework entwickelt wurde und der IDeRBlog-Chatbot muss lt. Vorgaben in den IDeRBlog-Lernplattform integriert werden. Zusätzlich wurde für die Kommunikation zwischen Client und Server das Spring-Websocket-Framework verwendet, um die permanente Verbindung zwischen Webfrontend und Webbackend aufrecht zu erhalten, damit die Echtzeitkommunikation zwischen BenutzerInnen und dem IDeRBlog-Chatbot ermöglicht wird. Im Abschnitt 4.3.2 wird die Websocket-Kommunikation zwischen Client und Server mit Codebeispielen detailliert erklärt.

3.2.6 Eingesetzte Entwicklungsumgebung IntelliJ

Eine **integrierte Entwicklungsumgebung** (englisch: Integrated Development Environment DIE), ist eine Programmierumgebung die in ein interaktives Anwendungsprogramm gepackt ist, die SoftwareentwicklerInnen in Entwicklungs- und Routinearbeiten unterstützen.

Eine solche integrierte Entwicklungsumgebung besteht typischerweise aus Editor, Compiler bzw. Interpreter, Debugger, Linker und einer grafischen Bedienoberfläche (GUI in Englisch: graphical user interface). Die Entwicklungsumgebung kann Teil einer vorhandenen Anwendung oder eine

eigenständige Anwendung sein. Eine integrierte Entwicklungsumgebung zeichnet sich aus durch eine optimierte grafische Schnittstelle mit intuitiver Bedienerführung, die das Erlernen der Funktionen und die Einarbeitung erleichtert. Eine entscheidende Rolle spielt die Bedienoberfläche, da alle Informationen hier zusammenlaufen. Die Bedienoberfläche sollte daher kontextorientiert sein und dem Bedienenden immer die von ihm benötigten Steuerelemente und Informationen für das gesamte Projekt zur Verfügung stellen. Somit bildet die Bedienoberfläche die Steuerzentrale für die Entwicklungswerkzeuge. Den BenutzerInnen stellen integrierte Entwicklungsumgebungen ein bedienungsfreundliches Framework für viele Programmiersprachen und Plattformen zur Verfügung, so unter anderem für Visual Basic, Java, Powerbuilder, .NET -Framework und Object Pascal, um nur einige hier zu nennen.

Zum Funktionsumfang und den Merkmalen einer integrierten Entwicklungsumgebung gehören die Informationen über den gesamten Projektumfang, eine Syntaxprüfung mit Ergänzungsmöglichkeiten, die Versionsverwaltung, die Erstellung von grafischen Bedienoberfläche, einen Profiler, damit die BenutzerInnen immer eine komplette Projektübersicht haben, die Verwaltung von Entwicklungsergebnissen, Debugger um den Programmablauf zu verfolgen, den Zugriff auf Teilergebnisse und einige weitere Funktionen (vgl. DATACOM Buchverlag GmbH 2019; o.S.).

IntelliJ IDEA ist eine integrierte Entwicklungsumgebung (IDE) oder spezielle Programmierumgebung, die hauptsächlich für Java entwickelt wurde. Für die Entwicklung von Programmen wird insbesondere diese Umgebung verwendet. Von der Firma JetBrains wurde das Programm entwickelt, welches offiziell IntelliJ benannt wurde. Erhältlich ist es in zwei Versionen: der Community Edition, die von Apache 2.0 lizenziert wird und einer kommerziellen Edition, der als Ultimate Edition bekannt ist. Beide Editionen werden zum Erstellen von Software verwendet. Was IntelliJ IDEA so sehr von seinen Pendants unterscheidet, ist die Flexibilität, Bedienungsfreundlichkeit und das solide Design des Programms.

Im Jahr 2001 wurde IntelliJ erstmals veröffentlicht und es zeichnet sich durch Funktionen wie erweiterte Code-Navigation und die Möglichkeit zum Umgestalten von Codes aus, was es sehr beliebt macht. Es wurde sogar als bestes Programmiertool auf Java-Basis gewählt. Dabei wurden etablierte Tools wie Eclipse, NetBeans und JDeveloper unterbunden. Von Google wurde die Open-Source-Entwicklungsumgebung für Android, basiert ebenfalls auf IntelliJ IDEA, im Jahr 2014 veröffentlicht. Die IDE unterstützt viele andere Programmiersprachen wie Python, Lua und Scala.

Es wird als eines der besten Programmiertools auf Java-Basis betrachtet. Hauptgründe sind dafür die Hilfsfunktionen, die einfache Bedienung und die gute Gestaltung der von ihm erstellten Programme. Es verfügt auch über erweiterte Fehlerprüfungsfunktionen, die eine einfachere und schnellere Fehlerprüfung ermöglichen (vgl. Techopedia 2019, o.S.).

Die Software IntelliJ wird für Windows, macos oder für Linux unter dem Link https://www.jetbrains.com/idea/download/#section=windows (abgerufen am 25.03.2019) als Download bereitgestellt. Die Ultimate Version wird als kostenlose Testversion für die Web- und Unternehmensentwicklung angeboten und die Community Version für die JVM und Android-Entwicklung wird kostenlos als Open Source zur Verfügung gestellt. Auf der Webseite https://www.jetbrains.com/idea/ (besucht am 25.03.2019) werden die Eigenschaften und Besonderheiten detailliert beschrieben, ebenso wird Unterstützung bei der Installation und bei möglichen Problemstellungen angeboten.

Im Rahmen dieser vorliegenden Masterarbeit wurde mit IntelliJ mit der Bildungslizenz der Technischen Universität Graz gearbeitet. IntelliJ wurde anderen Entwicklungsumgebungen vorgezogen, weil sie alle bereits genannten Technologien (HTML, CSS, JavaScript, Java und Java Spring-Framework), die für den IDeRBlog-Chatbot-Prototypen eingesetzt wurden, am besten unterstützt.

3.3 Eingesetzte Funktionen und Bibliotheken

In diesem Abschnitt werden die eingesetzten Bibliotheken beschrieben und erklärt, warum gerade diese Bibliotheken für die Entwicklung des Prototypen herangezogen wurden. Die Levenshtein Distanz, die eigentlich eine Funktion ist, wurde im nachfolgenden Abschnitt 3.3.1 beschrieben, weil diese zur Absichtserkennung herangezogen wird. Im Abschnitt 3.3.2 wird das LanguageTool angeführt, da es zur Fehlererkennung und Korrekturvorschläge eingesetzt wird und im Abschnitt 3.3.3 die Lombok-Bibliothek erläutert, weil diese die Programmierung in Java erleichtert.

3.3.1 Levenshtein Distanz-Funktion

Der russische Mathematiker Vladimir Levenshtein hat den Levenshtein-Algorithmus 1965 erfunden. Der Levenshtein-Algorithmus bzw. die Levenshtein-Distance ist ein Name-Matching-Verfahren und gehört bei vielen Datenbankprodukten zum Standardlieferumfang. Algorithmen zur Bestimmung der Levenshtein-Distance können als Programmcode kostenlos im Internet bezogen werden und sind recht einfach (vgl. Lisbach 2011, S.78).

Für diese Masterarbeit wurde eine Java-Implementierung des Levenshtein-Algorithmus über Apache-Maven bezogen. Durch die Levenshtein-Distance werden zwei Zeichenketten miteinander verglichen und gemeinsame Muster identifiziert, wenn diese Strings nicht exakt übereinstimmen. Daher wird es auch oft als einfaches Pattern-Matching-Verfahren bzw. auch als String-Comparison-Verfahren bezeichnet (vgl. Lisbach 2011, S.77).

Wo Schreibvariationen im Namen ausschließlich durch vollkommen zufällige Tippfehler zustande kommen und zwar im Sinne von unvorhersehbaren Vertauschungen, Auslassungen und Hinzufügungen einzelner Zeichen, ist die Levenshtein-Distance somit als Verfahren überall dort angebracht (vgl. Lisbach 2011, S.77).

Die Levenshtein Distanz-Funktion wird in dieser Masterarbeit zur Absichtserkennung aus der Nachricht verwendet.

3.3.2 LanguageTool

Die kostenlose Open-Source Applikation LanguageTool dient zur Überprüfung der Rechtschreibung, Grammatik und des Schreibstils von Texten. Im Hintergrund des Systems werden mit verschiedenen Regeln einzelne Wörter, Wortteile und ganze Sätze auf Übereinstimmung verglichen. Diese Regeln beschreiben in verschiedenen Kategorien typische Fehler. Wird ein Fehler erkannt, wird anschließend die entsprechende Textpassage markiert. Eine Fehlermeldung wird angezeigt, die je nach Anwendung zusätzliche Informationen zu dem Fehler ausgeben kann. Die NutzerInnen können so nicht nur den Fehler erkennen, sondern er kann auch eine Begründung für die Fehlerausgabe abrufen und, falls verfügbar, kann der Fehler auch direkt verbessert werden. Daher ist das Tool für verschiedene Gruppen bzw. NutzerInnen anwendbar. Es hilft sowohl nichtmuttersprachliche NutzerInnen als auch Muttersprachlern beim Erkennen von Rechtschreib- und Grammatikfehlern und unterstützt einen ansprechenden Schreibstil. Derzeit unterstützt das LanguageTool 31 Sprachen. Darunter sind zahlreiche Sprachen, sowie verschiedene Dialekte, zu finden (vgl. Keller 2018, o.S.).

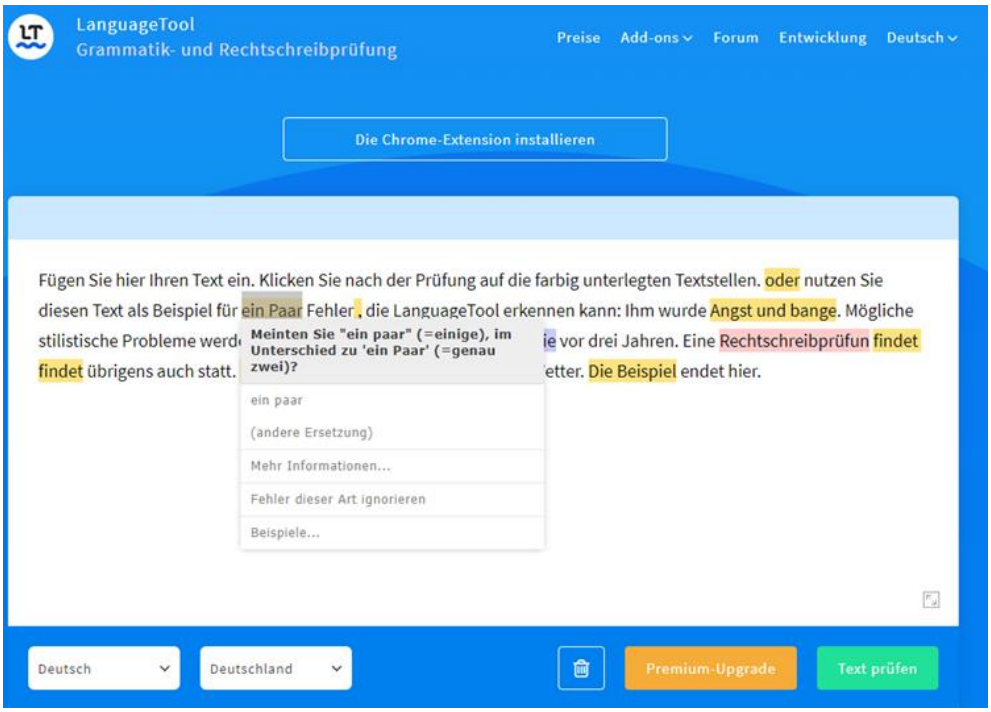

Abbildung 13: LanguageTool Webseite (LanguageTooler GmbH, 2019)

Mehr als 2500 Fehler können derzeit in deutschsprachigen Texten durch das LanguageTool erkannt werden. Wird der Fehler durch die Maus angesteuert, erscheint eine Fehlermeldung durch eine Anzeige (siehe Abbildung 13: LanguageTool Webseite (LanguageTooler GmbH, 2019)

Durch das LanguageTool werden Fehler der

- Grammatik,
- Groß- und Kleinschreibung,
- Zusammen- und Getrenntschreibung,
- Zeichensetzung (z.B. Beistrichfehler),
- Umgangssprache
- und Tippfehler erkannt.

Ebenso wird eine Datumsprüfung durchgeführt und es wird in fremdsprachigen Texten auf Fehler bzw. Wörter hingewiesen, die in Deutsch und einer Fremdsprache gleich oder ähnlich klingen oder geschrieben werden, aber etwas anderes bedeuten. Eine Prüfung auf leichte Sprache wird

vom LanguageTool ebenso unterstützt (vgl. LanguageTooler GmbH 2019, o.S.).

Die Entwicklung des LanguageTools zielt darauf ab, den NutzerInnen ein Instrument zu bieten, um mit weniger Fehlern Dokumente zu verfassen. Das Ziel ist möglichst alle Fehler in Texten durch das Tool zu finden und die Fehler dann auszugeben. Die Fehler sollten möglichst direkt erkannt und ausgegeben werden, damit der NutzerInnen interaktiv mit dem Tool arbeiten können und damit die NutzerInnen diese Fehler direkt verbessern können. Eine falsche Fehlerausgabe sollte zusätzlich ausgeschlossen werden, um die Glaubwürdigkeit an das Tool nicht einzuschränken und keine Fehler auszugeben, die keine Fehler sind. Das Tool soll außerdem für die verschiedenen Zwecke und Zielgruppen angepasst werden können.

Aufgrund verschiedener vordefinierter Regeln im Hintergrund des Tools erfolgt die Überprüfung nach Fehlern. Jedem Wort muss in der entsprechenden Textpassage das entsprechende Part-of-Speech Tag hinzugefügt werden und jeder Satz muss entsprechend in einzelne Chunks aufgesplittet werden, um Fehler erkennen zu können. Somit können Wortkombinationen, einzelne Wörter und sogar Sätze mit den Regeln verglichen werden und sobald eine Übereinstimmung besteht werden die Fehler angezeigt. Verschiedene Hilfsmittel werden zur Überprüfung des Textes benötigt. Diese Hilfsmittel sind ein Chunker und ein sogenannter Part-of-Speech Tagger. Man versteht in der Computerlinguistik unter "Tagging" allgemein die Annotation von Korpora mit linguistischen Informationen. Die üblichen Part-of-Speech Tags sind Verben, Adjektive oder Adverben und Nomen. Beispielsweise können Nomen zusätzlich als Singular oder Plural definiert werden.

Es ist oft schwierig für den Algorithmus den passenden Tag zu erkennen, da ein Wort oft verschiedene POS-Tags haben kann. Daher gibt es einen Disambiguator, der versucht durch verschiedene vordefinierte Regeln auf Basis der Wortstellung, der Semantik oder aufgrund von Häufigkeiten das entsprechende POS-Tag auszuwählen. Im Gegensatz zum POS-Tagging wird beim Chunking einer Sequenz von Wörtern ein Tag zugeordnet. Typische

Tags können beispielsweise eine Verbphrase oder eine Nominalphrase sein. Die Erkennung der Fehler wird durch den Ansatz der regelbasierten Überprüfung unterstützt. Vorher müssen die Regeln dazu im System eingetragen werden. Nachteilig ist, dass Fehler, die nicht vorher definiert wurden, daher nicht erkannt werden können. Es ist sehr schwierig bei einer großen Häufigkeit an möglichen Fehlern ein komplettes Regelset zu definieren. Das Regelset kann jedoch Schritt für Schritt aufgebaut werden und von den Nutzern aufgrund einer relativ verständlichen Syntax erweitert werden. Fehlermeldungen können außerdem integriert werden, die dem NutzerInnen die Möglichkeit geben, dessen Fehler besser zu verstehen und optional zu verbessern.

Das Regeln mit einer einfachen XML Syntax geschrieben werden können, ist ein weiterer Aspekt des LanguageTools. Die NutzerInnen selbst können Regeln schreiben und diese implementieren oder auch an den EntwicklerInnen des Tools weitergeben. Die Muster zur Fehlererkennung können mit Hilfe eines Regel-Editors erstellt werden. Der Regeleditor ist unter dem Link https://community.languagetool.org/ruleEditor2/index?lang=de (zuletzt aufgerufen am 25.03.2019) zu finden (vgl. Keller 2018, o.S.).

Zur Zeit der Entwicklung des IDeRBlog-Chatbots wurden keine passenden Alternativen zum LanguageTool API-Server, der eine Rest-API für die Korrektur der Rechtschreibfehler in der deutschen Sprache zur Verfügung stellt, gefunden. Daher wurde in der vorliegenden Masterarbeit das LanguageTool zur Schreibfehlererkennung und für die Lieferung von Korrekturvorschläge für den IDeRBlog-Chatbots herangezogen. Die Anwendung im Prototypen wird im Abschnitt 4.3.5 erläutert.

3.3.3 LOMBOK

Lombok bzw. Projekt LOMBOK ist eine Java-Bibliothek, die sich automatisch in die Entwicklungsumgebungen und Editoren eingebunden werden kann. Die Bibliothek unterstützt das Programmieren in Java (vgl. The Project Lombok

Authors 2009-2019, o.S.). Lombok generiert Codes, die auf Anmerkungen bzw. Annotationen basieren. Dies macht die Java-Entwicklung viel einfacher (vgl. Cornelißen et al. 2018, S.23). Auf der Webseite https://projectlombok.org/ (letzter Zugriff am 25.03.2019) kann die Bibliothek heruntergeladen werden und es gibt dort zahlreiche Informationen und Hilfestellungen. Eine Community unterstützt bei Fragen und bestimmten Themenstellungen.

In einen typischen Java-Projekt werden hunderte von Codezeilen geschrieben, die zum Definieren einfacher Datenklassen erforderlich sind. Das ist nicht ungewöhnlich für Java. Im Allgemeinen enthalten diesen Klassen mehrere Felder und für diese Felder werden Getter, Setter, Hash-Code- und Equals und ToString Methoden generiert. Lombok kann diese Klassen auf die erforderlichen Felder und auf eine einzige @Data-Annotation reduzieren.

- **@Data**
  Die am häufigsten verwendete Annotation im Project Lombok-Toolset ist wahrscheinlich die @Data-Annotation. Sie kombiniert die Funktionen von @Getter und @Setter, @ToString, @Equals- und HashCode. Die Annotation mit @Data und einer Klasse löst auch die Generierung von Lomboks Konstruktoren aus (vgl. Kimberlin 2015, o.S.).

Eine kurze Beschreibung der wichtigsten Annotationen, die im @Data-Annotation bereits erwähnt wurden, wird nachfolgend erläutert.

- **@Getter and @Setter**
  Mit der @Getter und @Setter-Annotation werden automatisch die Standard Getter und Setter-Methoden generiert. Diese generierten Methoden sind standardmäßig öffentlich, sofern die Zugriffsebene nicht explizit durch zusätzlichen Parameter (private, public, protected, package) definiert wurde (vgl. The Project Lombok Authors 2009-2019, o.S.).

- **@ToString**
  Eine Implementierung der toString-Methode wird durch diese Annotation generiert. Standardmäßig werden in die Ausgabe der Methode alle nicht statischen Felder in Name-Wert-Paaren aufgenommen. Die Einbeziehung der Eigenschaftsnamen in die Ausgabe kann unterdrückt werden, indem der Annotation-Parameter *includeFieldNames auf false* gesetzt wird (falls gewünscht).

- **@EqualsAndHashCode**
  Auf Klassenebene führt diese Annotation dazu, dass Lombok sowohl *equals-* als auch *hashCode-Methoden* generiert, da beide durch den *hashCode-Vertrag* untrennbar miteinander verbunden sind. Jedes Feld in der Klasse wird standardmäßig, das nicht statisch oder transient ist, von beiden Methoden berücksichtigt. Der Parameter *exclude* wird ähnlich wie bei *@ ToString* bereitgestellt, um zu verhindern, dass Felder in die generierte Logik aufgenommen werden.

Es gibt auch in Lombok eine Reihe von Annotationen, die eine feinere Kontrolle des Verhaltens und der Struktur einer Klasse ermöglichen (vgl. Kimberlin 2015, o.S.), die aber in diesem Projekt nicht angewendet wurden.

Wie bereits angedeutet, erleichtert Lombok die Programmierung in Java. Lombok wurde im Webbackend bei der Entwicklung des IDeRBlog-Chatbots angewendet, weil dadurch die Programmierzeilen des entwickelten Codes stark reduziert werden und ein späteres Refactoring (Umprogrammierung) des Codes leichter wird.

4 UMSETZUNG DES PROTOTYPEN

Dieses Kapitel widmet sich der Umsetzung des Prototypens. Abschnitt 4.1 widmet sich der IDeRBlog-Chatbot-Architektur und im Abschnitt 4.2 wird die Implementation des Clients bzw. des Webfrontends aufgezeigt. Das Design wird mit Abbildungen anschaulich dargestellt, um einen Einblick in das Aussehen des IDeRBlog-Chatbots zu geben. Des Weiteren wird die Implementation vom Backend beschrieben. Für eine bessere Nachvollziehbarkeit wurden verschiedene Abbildungen hinzugezogen.

4.1 Architektur

Der IDeRBot-Chatbot ist ein selbst programmiertes Programm und stellt die Aufgabe dieser Masterarbeit dar. Durch das Lesen und Analysieren der Programmanforderungen wird daraus geschlossen, dass es nicht möglich ist, ein vorhandenes Framework zur Entwicklung eines Chatbots zu verwenden, der in den IDeRBlog integriert werden kann.

Für die Entwicklung der IDeRBlog-Lernplattform wurden die Technologien HTML, CSS, JavaScript und Java Spring Framework verwendet. Aus diesem Grund wurde auch der IDeRBlog-Chatbot mit den bereits genannten Technologien programmiert, sodass eine Integration auf die IDeRBlog-Lernplattform ermöglicht wird.

Die IDeRBot-Chatbot-Architektur basiert auf einer Client-Server-Architektur. Auf der Clientseite ist die Webfrontend- Bedienoberfläche, auf der die

BenutzerInnen Nachrichten an den Server schreiben und vom Server empfangen können. Die Client-Seite wurde vollständig getrennt vom Server entwickelt. Auf der Serverseite im Webbackend gibt es einen Message-Handler, der sich ebenso um das Empfangen und Senden von Nachrichten kümmert. Zusätzlich wurde im Backend die Programmierlogik der BenutzerInnen-Absichtserkennung, Extraktion der Entitäten (Schreibfehler, etc.) und Korrektur der Schreibfehler implementiert.

In der folgenden Abbildung 14 wird die Client-Serverarchitektur im Detail dargestellt. Daraus ist ersichtlich, wie eine Nachricht vom Client zur Server gesendet und die daraus resultierende Antwort wieder retourniert wird und daraus zuständigen Klassen und deren Verantwortung. Weitere Details werden in den anschließenden Abschnitten dieses Kapitels erläutert.

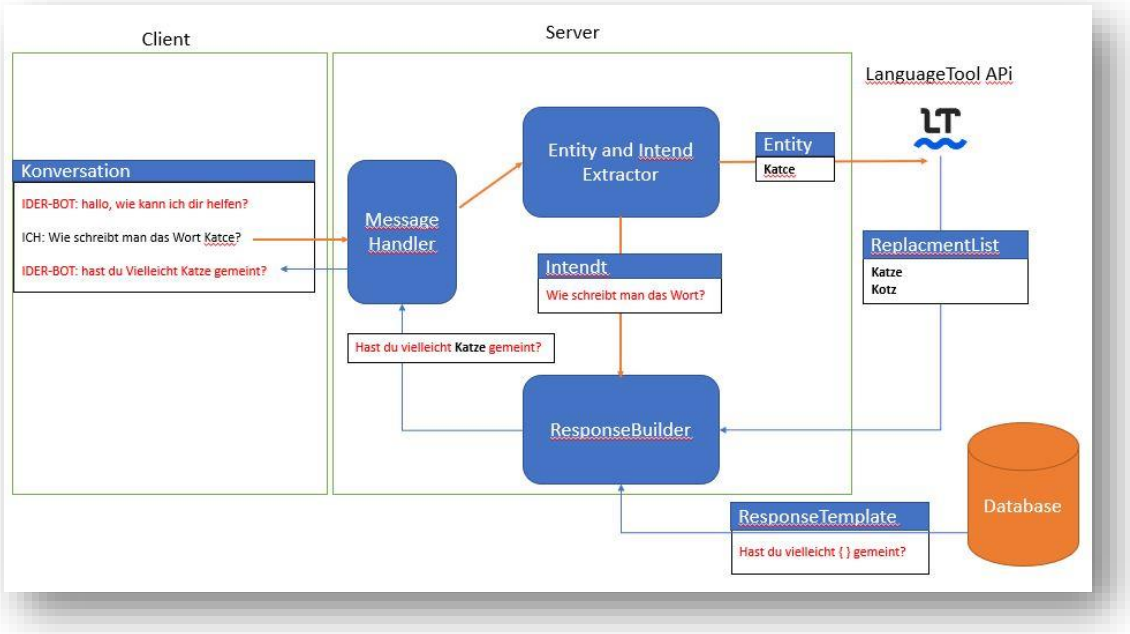

Abbildung 14: Chatbot Architektur

4.2 Implementation des Webfrontends

Die Implementation des Client wird in diesem Abschnitt ausführlich beschrieben. Im Abschnitt 4.2.1 wird das Design der Weboberfläche beschreiben und im Abschnitt 4.2.2 beschäftigt sich dem Verbindungsaufbau mit dem Server.

4.2.1 Design

Basierend auf den Anforderungen, dass der IDeRBlog-Chatbot von den meisten Heranwachsenden im Kindesalter verwendet werden wird, muss eine einfache Bedienoberfläche entworfen werden. Die Bedienoberfläche soll einfach gehalten werden und die Integration in den IDeRBlog soll möglichst mit wenig Aufwand erfolgen.

In der folgenden Abbildung 15 werden die Eingabe- und Ausgabemasken (Kommunikationsoberfläche), die für den Nachrichtenaustausch zwischen dem IDeRBlog-Chatbot und den BenutzerInnen verwendet werden, dargestellt.

Das Fenster mit der Bezeichnung Text Editor zeigt die Eingabemaske des zu korrigierenden Textes der IDeRBlog-Lernplattform während der Entwicklungsphase des IDeRBlog-Chatbots an. Auf der rechten Seite werden die Eingabemasken des Prototypens angezeigt.

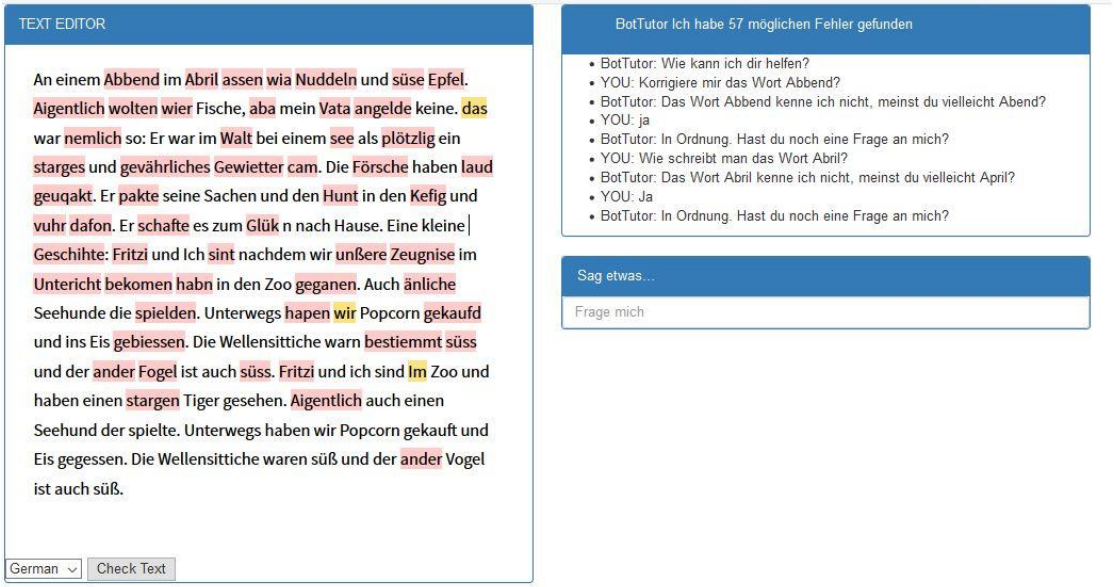

Abbildung 15: Kommunikationsoberfläche des IDeRBlog-Chatbots

Das Layout wurde durch Verwendung von Bootstrap Framework (sieh Kapitel 3.2.4) responsive gestaltet, um den Kindern die Möglichkeit zu geben, Konversationen nicht nur am Computer, sondern auch auf mobilen Geräten zu machen. Wie einfach es ist, für einen Prototypen die Ein- und Ausgabemasken mit Bootstrap zu erstellen, zeigt die anschließende Auflistung 3 des IDeRBlog-Chatbot-Webfrontends. Die Verwendung von vorhandenen CSS- und HTML-Vorlagen ermöglicht es in 12 Programmierzeilen Code die Masken zu definieren, gestalten und responsive für verschiedene Auflösungen zu erstellen.

```
01 <div class="col-lg-6">
02     <div class="panel panel-primary">
03         <div class="panel-heading">BotTutor<ul
                id="botHeader"></ul></div>
04         <div><ul id="ulMessages"></ul></div>
05         </div>
06         <div class="panel panel-primary">
07             <div class="panel-heading">Sag etwas...</div>
08         <input id="textInput" placeholder="Frage mich" class="form-
  control" size="81" name="message" value="Wie schreibt man das Wort
  Katce?" type="text" onkeyup="sendWithEnter(event)"/>
09     </div>
10 </div>
```

Auflistung 3: IDeRBlog-Chatbot-Webfrontend

4.2.2 Aufbau der Serververbindung

Die Kommunikation zwischen Client und Server erfolgt durch Verwendung des Websocket-Protokolls. Ein Websocket-Objekt wird erstellt und es wird automatisch versucht eine permanente Websocket-Verbindung zum Server aufzubauen.

Der Konstruktor vom Websocket-Objekt benötigt den URL, auf die der Websocket-Server antwortet. Durch die Websocket- und Open-Methode wird die Verbindung geöffnet und zeigt an, dass die Verbindung zum Senden und Empfangen von Nachrichten bereit ist (siehe Auflistung 4: Verbindungsaufbau). Die gesamten Nachrichten, die zwischen Client zum Server ausgetauscht werden, werden als JSON-String übermittelt.

JSON ist ein Dateiformat, mit dem eine Datenübertragung zwischen Client und Server möglich ist. Mithilfe diese Dateiformat kann man strukturierte Daten zwischen Client und Server übertragen und auf Basis dieser Daten, Inhalte generieren (vgl. Ackermann 2018, S.510f.). Die geschweiften Klammern, über die einzelne Objekte definiert werden, ist ein wesentliches Kennzeichen des JSON-Formats. In Anführungszeichen werden die Objekteigenschaften geschrieben und durch einen Doppelpunkt von dem jeweiligen Wert getrennt (vgl. Ackermann 2018, S.517). Die anschließende Auflistung 4 zeigt die Nachricht, die in der Variablen *var* gespeichert ist, als JSON-Format an.

```
01 var wsUri = "ws://localhost:8080/iderbot";
02 var websocket = new WebSocket(wsUri);
03 var parameter = false;
04 websocket.onopen = function (evt) {
05    generateUniqId();
06    var msg = {
07        type: "FIRST_CONNECT",
08        messageText: "HELLO FROM CLIENT",
09        clientId:   client_id
10    };
11
12    websocket.send(JSON.stringify(msg));
13 }
```

Auflistung 4: Verbindungsaufbau

Die Datenstruktur die als Nachricht im JSON-Format gespeichert wird, ist wie folgt in der Tabelle 5 definiert:

MESSAGE_TYPE	FIRST_CONNECTION MESSAGE END_CONNECTION
MESSAGE TEXT	Nachrichtentext
ACTIONS	weiterer Parameter
CLIENT ID	eindeutige ID des Clients

Tabelle 5: Beschreibung der Nachrichteneigenschaften

Nun folgt eine kurze Beschreibung über den jeweiligen Nachrichtentyp.

MESSAGE_TYPE: ist ein **Nachrichtentyp,** der zwischen Client und Server ausgetauscht wird. Insgesamt gibt es drei Nachrichtentypen. Diese sind:

- FIRST_CONNECTION: Wird bei erstmaliger Herstellung der Verbindung vom Client zum Server versendet.
- MESSAGE: Ist die tatsächliche Textnachricht, welche zwischen Client und Server ausgetauscht wird.
- END_CONNECTION: Wird beim Schließen der Websocket-Verbindung versendet.

MESSAGE TEXT: Bezeichnet den **Nachrichtentext** in seiner Gesamtlänge, die vom Server und Client in Entitäten und Absichten zerlegt wird.

ACTIONS: Ist eine **Liste mit Parametern**, die beliebig erweitert werden kann, die im Zuge einer Konversation verwendet wird.

CLIENT ID: Ist die **eindeutige Client-ID**, die jeweils zwischen Client und Server bei jeder Nachricht versendet wird, um eine eindeutige Identifizierung zu gewährleisten und bleibt während der gesamten Konversation bis zum nächsten Verbindungsaufbau eindeutig.

4.3 Implementation vom Backend

Dieser Abschnitt beschäftigt sich mit der Implementation des Backend des IDeRBlog-Chatbots. Durch Codebeispiele wird sowohl die Konfiguration der Web-Applikation als auch die WebSocketverbindung veranschaulicht. Ebenso wird erklärt wie die Absichten definiert und erkannt werden. Der Extraktion von Schreibfehlern (Entitäten) ist auch ein weiterer Abschnitt gewidmet. Abschließend wurde die Generierung der Korrekturvorschläge und der Antworten aufgezeigt.

4.3.1 Konfiguration des Projektes

Um das Backend für die Web-Applikation im Spring Framework zu entwickeln, wird eine Java Spring-Web Applikation mithilfe des Build-Management-Tools Maven konfiguriert. Zur Konfiguration der Anwendung verwendet Maven eine POM-Dateien. Zu den Konfigurationen, die im POM angegeben werden können, gehören die Projektabhängigkeiten, die auszuführenden Plugins, die Build-Profile und weitere Einstellungen. Weitere Informationen bezüglich Projektversion, Beschreibung, EntwicklerInnen, Mailinglisten, usw. können ebenfalls in dieser POM-Datei definiert werden (vgl. The Apache Software Foundation 2019, o.S.).

Zur Zeit der Entwicklung wurde die Apache Maven Version v3.6.0., Java Version 1.8 und die Spring Boot Version 1.5.9.RELEASE eingesetzt. Die

Entwicklung wurde in der IntelliJ-IDEA gemacht. Weitere verwendete Abhängigkeiten und Plugins sind zur besseren Nachvollziehbarkeit folgend in der Auflistung 5 der POM-Konfigurationsdatei veranschaulicht dargestellt.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
03     <modelVersion>4.0.0</modelVersion>
04     <groupId>at.tugraz</groupId>
05     <artifactId>IDeRBlogChatbot</artifactId>
06     <version>0.0.1-SNAPSHOT</version>
07     <packaging>jar</packaging>
08
09     <name> IDeRBlogChatbot </name>
10     <description> Chatbot für Rechtschreibfehlerkorrektur
</description>
11
12     <parent>
13         <groupId>org.springframework.boot</groupId>
14         <artifactId>spring-boot-starter-parent</artifactId>
15         <version>1.5.9.RELEASE</version>
16         <relativePath/>
17     </parent>
18
19     <properties>
20         <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
21         <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
22         <java.version>1.8</java.version>
23     </properties>
24
25     <dependencies>
26         <dependency>
27             <groupId>org.springframework.boot</groupId>
28             <artifactId>spring-boot-starter-data-jpa</artifactId>
29         </dependency>
30         <dependency>
31             <groupId>org.springframework.boot</groupId>
32             <artifactId>spring-boot-starter-thymeleaf</artifactId>
33         </dependency>
34         <dependency>
35             <groupId>org.springframework.boot</groupId>
36             <artifactId>spring-boot-starter-websocket</artifactId>
37         </dependency>
38         <dependency>
39             <groupId>org.projectlombok</groupId>
40             <artifactId>lombok</artifactId>
41             <optional>true</optional>
42         </dependency>
43         <dependency>
44             <groupId>org.springframework.boot</groupId>
```

```
45              <artifactId>spring-boot-starter-test</artifactId>
46              <scope>test</scope>
47          </dependency>
48          <dependency>
49              <groupId>com.h2database</groupId>
50              <artifactId>h2</artifactId>
51          </dependency>
52          <!-- for switching to MySql in production mode
53          <dependency>
54              <groupId>mysql</groupId>
55              <artifactId>mysql-connector-java</artifactId>
56              <scope>runtime</scope>
57          </dependency>
58          -->
59          <dependency>
60              <groupId>org.languagetool</groupId>
61              <artifactId>languagetool-core</artifactId>
62              <version>LATEST</version>
63          </dependency>
64          <dependency>
65              <groupId>org.languagetool</groupId>
66              <artifactId>language-de</artifactId>
67              <version>LATEST</version>
68          </dependency>
69          <dependency>
70              <groupId>com.google.code.gson</groupId>
71              <artifactId>gson</artifactId>
72              <version>2.8.2</version>
73          </dependency>
74          <dependency>
75              <groupId>info.debatty</groupId>
76              <artifactId>java-string-similarity</artifactId>
77              <version>RELEASE</version>
78          </dependency>
79      </dependencies>
80
81      <build>
82          <plugins>
83              <plugin>
84                  <groupId>org.springframework.boot</groupId>
85                  <artifactId>spring-boot-maven-plugin</artifactId>
86              </plugin>
87          </plugins>

88      </build>
```

Auflistung 5: POM-Konfigurationsdatei

Für den Datenbankzugriff wurde in dieser Arbeit Java Persistence API eingesetzt und es wurde für die Entwicklung des Prototyps die H2-in-Memory-Datenbank verwendet. Die nötige Konfiguration wurde in der application.properties-Konfigurationsdatei erstellt (siehe Auflistung 6).

```
01 spring.datasource.url=jdbc:h2:file:~/test
02 spring.datasource.username=sa
03 spring.datasource.password=
04 spring.datasource.driver-class-name=org.h2.Driver
05 spring.h2.console.enabled=true
06 spring.h2.console.path=/h2
07 spring.jpa.hibernate.ddl-auto= create-drop
```

Auflistung 6: Datenbankkonfiguration in application.properties Datei

Zur Bereitstellung der ausführbaren WAR-Datei wird der Apache Tomcat-Webserver in der Version Apache Tomcat/8.5.23 verwendet.

4.3.2 WebSocket-Verbindung

Um Nachrichten zu empfangen, muss im Backend eine permanente Socket-Verbindung für alle verbundenen Clients zu Verfügung stehen. Für diese Umsetzung wurde das WebSocket Protokoll von Spring Framework (vgl. spring.io 2019, o.S.) verwendet.

In der anschließenden Auflistung 7 wird der Einsatz des TextWebSocketHandlers in der Klasse Message-Handler detailliert veranschaulicht.

In der MessageHandler Klasse die vom TextWebSocketHandler abgeleitet wird, werden in der Methode handleTextMessage() alle Nachrichten, die von Clients ankommen, verarbeitet. Jeder Client wird nach dem Verbindungsaufbau mit einer Client-ID eindeutig identifiziert. Diese ID wird auch für die Identifikation der offenen Konversationen verwendet.

```
01 public class MessageHandler extends TextWebSocketHandler {
02     Set<String> clientIds = new HashSet<>();     List<Conversation>
           conversationList = new ArrayList<>();
03     JLanguageTool jLanguageTool = new JLanguageTool(new
04     GermanyGerman());.
05     @Override
06     protected void handleTextMessage(WebSocketSession session,
07           TextMessage textMessage) throws Exception {
08            Message obj = new Message();
```

```
09          Gson gson = new Gson();
10          obj = gson.fromJson(textMessage.getPayload(),
11            Message.class);
12          Conversation conversation = null;
13          if (obj.getType().equals("FIRST_CONNECT")) {
14                System.out.println("CLIENT CONNECTED: " +
15                  textMessage.getPayload().toString());
16                obj.setType("SERVER-MESSAGE");
17                obj.setMessageText(sendRandomGreetingToClient());
18                clientIds.add(obj.getClientId());
19                session.sendMessage(new
20                  TextMessage(gson.toJson(obj)));
21                conversation = new Conversation();
22                conversation.setClientId(obj.getClientId());
23                conversationList.add(conversation)
```

Auflistung 7: Klasse Message Handler

4.3.3 Absichten (Intents) definieren

Unter der Berücksichtigung der Anforderungen für den Prototypen mussten folgende Beispielszenarien einer menschenähnlichen Kommunikation überlegt werden. In diesem Beispielszenarien, es wird angenommen, dass die BenutzerInnen einfache Fragen an den IDeRBlog-Chatbot stellen wird. In der Abbildung 16 wird beispielhaft eine mögliche einfache Frage des Benutzers und die darauffolgende Antwort des IDeRChatbot dargestellt.

In der Abbildung 16 wird dem IDeRBlog-Chatbot die Frage „Hallo, wie geht es dir?“ von BenutzerInnen (grün hinterlegt) gestellt. Der IDeRBlog-Chatbot antwortet daraufhin „Danke, mir geht es gut!“. Damit der IDeRBlog-Chatbot eine einfache Konversation führen kann, wurden zunächst die möglichen Ausdrücke und Antworten für die Standartbegrüßungsabsicht (DefaultWelcomeIntent) überlegt. Nachfolgend werden diese Überlegungen zu den Standart-Begrüßungen in der Abbildung 17 dargestellt.

Abbildung 16: Beispielannahme einer einfachen Konversation

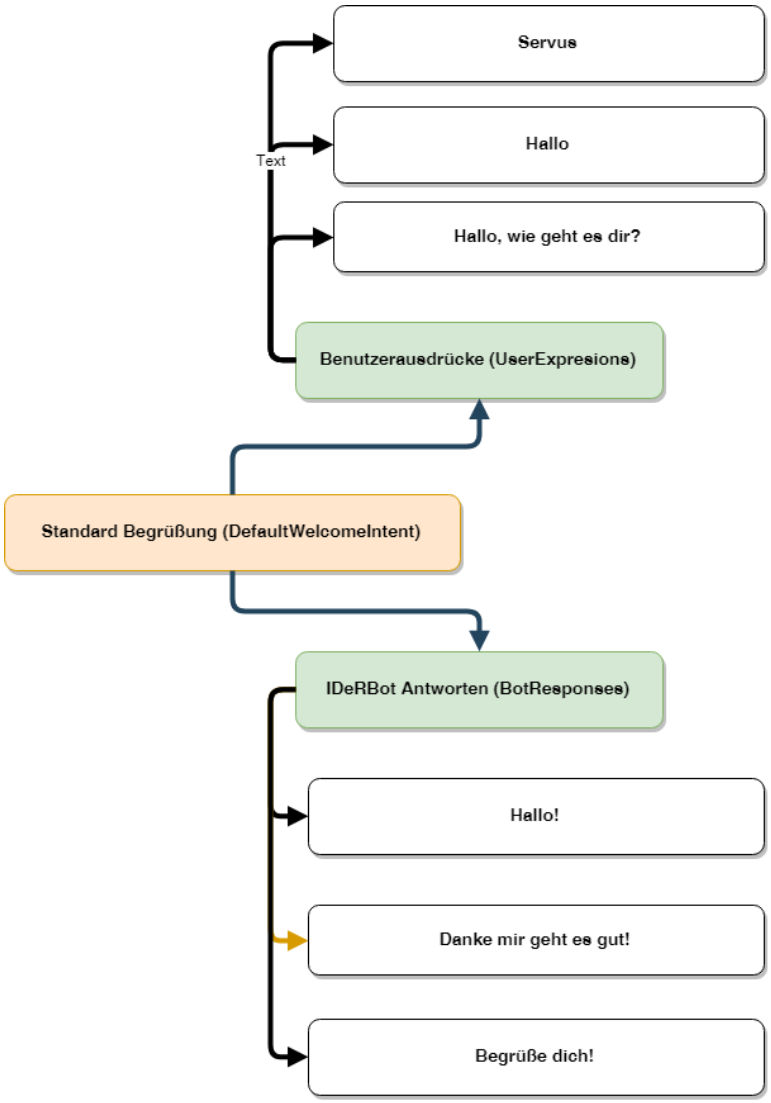

Abbildung 17: Überlegungen zu Standard-Begrüßungsabsicht

Die Hauptanforderung des IDeRBlog-Chatbots ist, Schreibfehler von einzelnen Wörtern zu erkennen und Korrekturvorschläge bzw. Korrekturhinweise den BenutzerInnen zu liefern.

Die Fragestellung des Benutzers könnte zum Beispiel **„Korrigiere mir das Wort Katce?"** sein, wie in der Abbildung 18 dargestellt wird. Dies ist eine von vielen möglichen Fragen zur Rechtschreibkorrektur, die an den IDeRBlog-Chatbot gestellt werden können.

Anhand dieses Beispiels in der Abbildung 18 wird die Rechtschreibfehlerabsicht (MisSpellingIntent) definiert, die in der Abbildung 19 dargestellt wird.

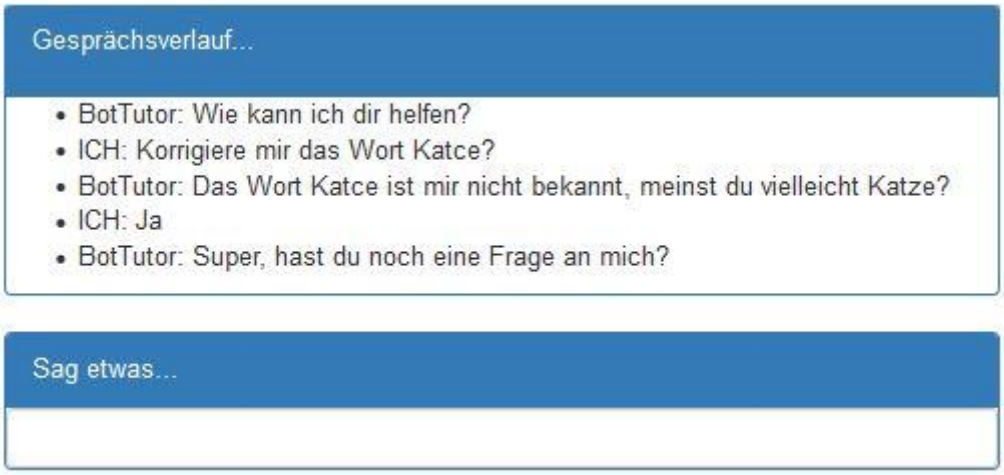

Abbildung 18: Beispielannahme einer komplexen Konversation

Das Wort „Katce" enthält hier einen Rechtschreibfehler, den der IDeRBlog-Chatbot erkennen muss. Aus dieser Fragestellung können längere Konversationen mit weiteren Verläufen entstehen. Die Entstehung dieses Konversationsverlaufs erforderte zusätzliche Überlegungen. Aus diesem Grund wurden Slots und Aktionen definiert.

- Slots dienen als Platzhalter für das falsch geschriebene Wort (MisSpelling) und die Korrektur (Replacement), die in der Antwort als Nachricht an die BenutzerInnen retourniert wird. Zusätzlich wird

durch weitere Hinweise bzw. Gruppenzuweisungen („Meinst du vielleicht Katze, das Tier?“) das richtige Wort von den BenutzerInnen ermittelt.

- Aktionen sind Parameter, die an die BenutzerInnen vom IDeRBlog-Chatbot gesendet werden. Die Zustimmung eines Korrekturvorschlags ist eine Aktion, die die BenutzerInnen mit „Ja“ zustimmen oder mit „Nein“ ablehnen müssen.

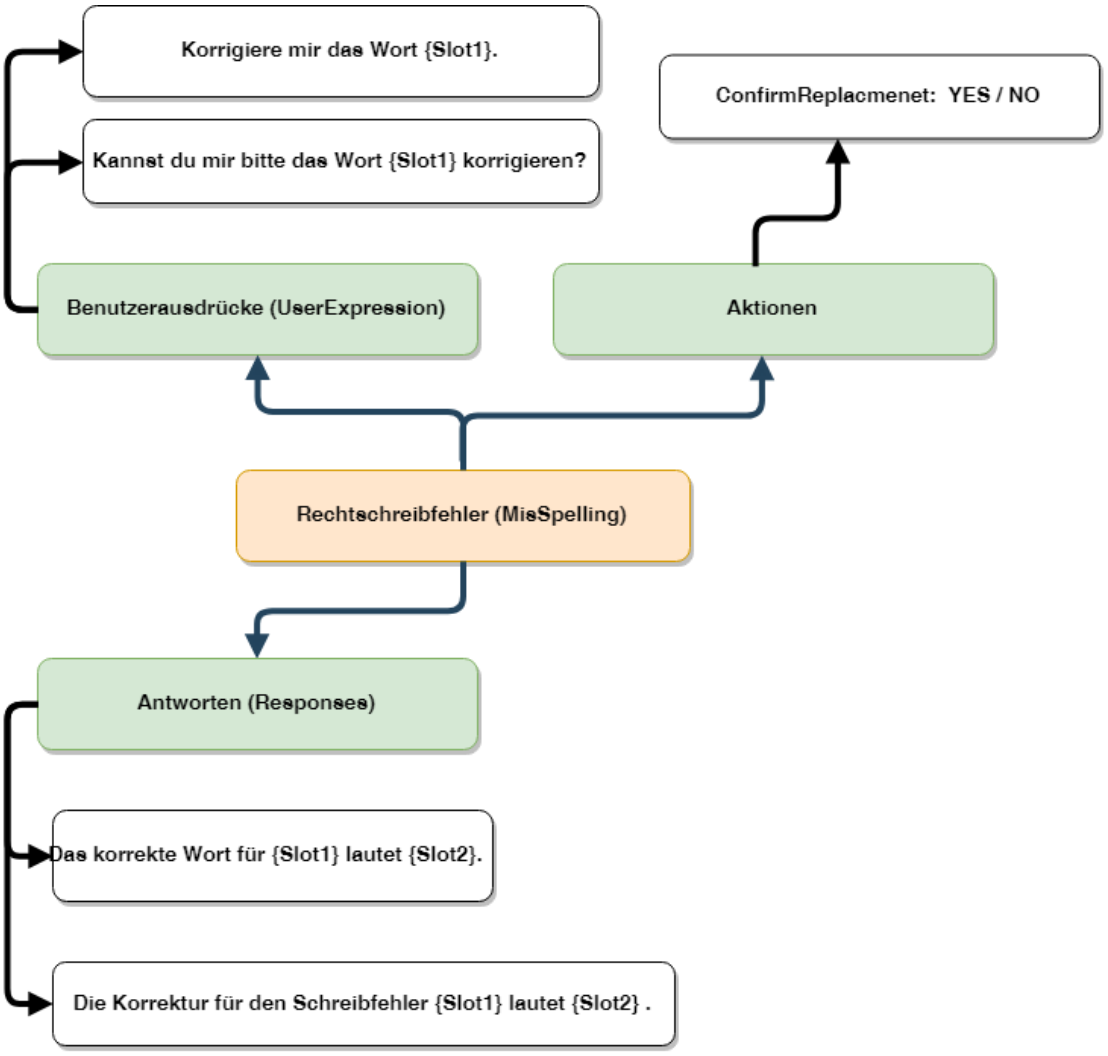

Abbildung 19: Definition der Absicht (Intent) für den Rechtschreibfehler

Die Umsetzung im Backend wurde in der Klasse Intent realisiert, die hier in der folgenden Auflistung 8 veranschaulicht wird.

Anwendung im Prototypen:

```java
01 package at.tugraz.chatbot.model;
02 import lombok.Getter;
03 import lombok.Setter;
04 import javax.persistence.*;
05 import java.util.HashMap;
06 import java.util.Map;
07 import java.util.Set;
08
09 @Setter
10 @Getter
11 @Entity
12 public class Intent {
13    @Id
14    @Column(name = "id", unique = true, nullable = false)
15    @GeneratedValue(strategy = GenerationType.AUTO)
16    Long id;
17    private String intentName;
18    int used;
19    @Transient
20    Map<String, String> aktions = new HashMap<>();
21    @OneToMany(mappedBy = "intent")
22    private Set<TrainingPhrases> trainingPhrases;
23    @OneToMany(mappedBy = "intent", fetch = FetchType.EAGER)
24    private Set<Responses> responses;
25 }
```

Auflistung 8: Klasse Intent

Eine Absicht (Intent) zu erkennen, ist der wesentlichste Bestandteil des IDeRBlog-Chatbots. Für die Erkennung der Absicht wurde der Levenshtein-Distanz Algorithmus verwendet, der in der Java-Bibliothek (java-string-library) enthalten ist.

Die java-string-library „Bibliothek" unterliegt den Lizenzbedingungen der MIT-Lizenz https://opensource.org/licenses/MIT (besucht am 06.04.2019), diese ist unter https://github.com/tdebatty/java-string-similarity (besucht am 06.04.2019) abrufbar. Die Einbindung der Bibliothek in das Projekt erfolgte über die POM.XML-Datei (siehe Auflistung 5).

Anwendung im Prototypen

```java
01 public Intent searchForIntentInUserExpresion(String message,
                                                List<TrainingPhrases> trainingPhrases) {
02    Double min = Double.MAX_VALUE;
03    Levenshtein levenshtein = new Levenshtein();
04    Intent intent = null;
05    for (TrainingPhrases trainingPhrase : trainingPhrases) {
```

```
06      if (levenshtein.distance(message, trainingPhrase.getUserExpression()) <
        min) {
07          min = levenshtein.distance(message,
                                       trainingPhrase.getUserExpression());
08          match = trainingPhrase.getUserExpression();
09          intent = trainingPhrase.getIntent();
10          }
11      }
12  return intent
```

Auflistung 9: Absichtserkennung durch den Levenshtein-Distanz Algorithmus

Die Auflistung 9 zeigt die Erkennung der Absicht durch den Levenshtein-Distanz Algorithmus. Die Funktion *distance()* nimmt die Nachricht als Parameter auf und durchläuft die Liste mit Benutzerausdrücken aus den Testdaten (TrainingPhrases). Aus den Testdaten wird der Satz (TrainingPhrases) mit der niedrigsten Distanz ausgewählt und daraus wird die Absicht (Intent) erkannt. Eine genauere Beschreibung der Funktionsweise des Levenshtein Distanz Algorithmus ist im Abschnitt 3.3.1 zu finden.

4.3.4 Extrahierung von Schreibfehlern

Für die Extrahierung der Wörtern, die einen Rechtschreibfehler enthalten wurde wie folgt vorgegangen. Nachdem die Absicht (Intent) von der Nachricht durch den Levenshtein-Distanz Algorithmus erkannt wurde, muss als nächstes der Schreibfehler aus der Nachricht extrahiert werden.

Zuerst wird die Nachricht in einzelne Wörter zerlegt. Die einzelnen Wörter aus der Nachricht, die eine Übereinstimmung mit den Wörtern aus dem Intent haben, werden ignoriert. Die übrig geblieben Wörter in der Nachricht, die im Intent gar nicht vorkommen, werden extrahiert. Diese extrahierten Wörter sind Kandidaten die einen Rechtschreibfehler enthalten könnten.

In der Abbildung 20 wird die Extraktion eines Rechtschreibfehlers (Entität) des Wortes „Katce" in roter Farbe dargestellt. Das extrahierte Wort, welches eventuell einen Schreibfehler enthalten könnte, muss im nächsten Schritt

einer Korrekturroutine unterzogen werden, die im nächsten Abschnitt genau erläutert wird.

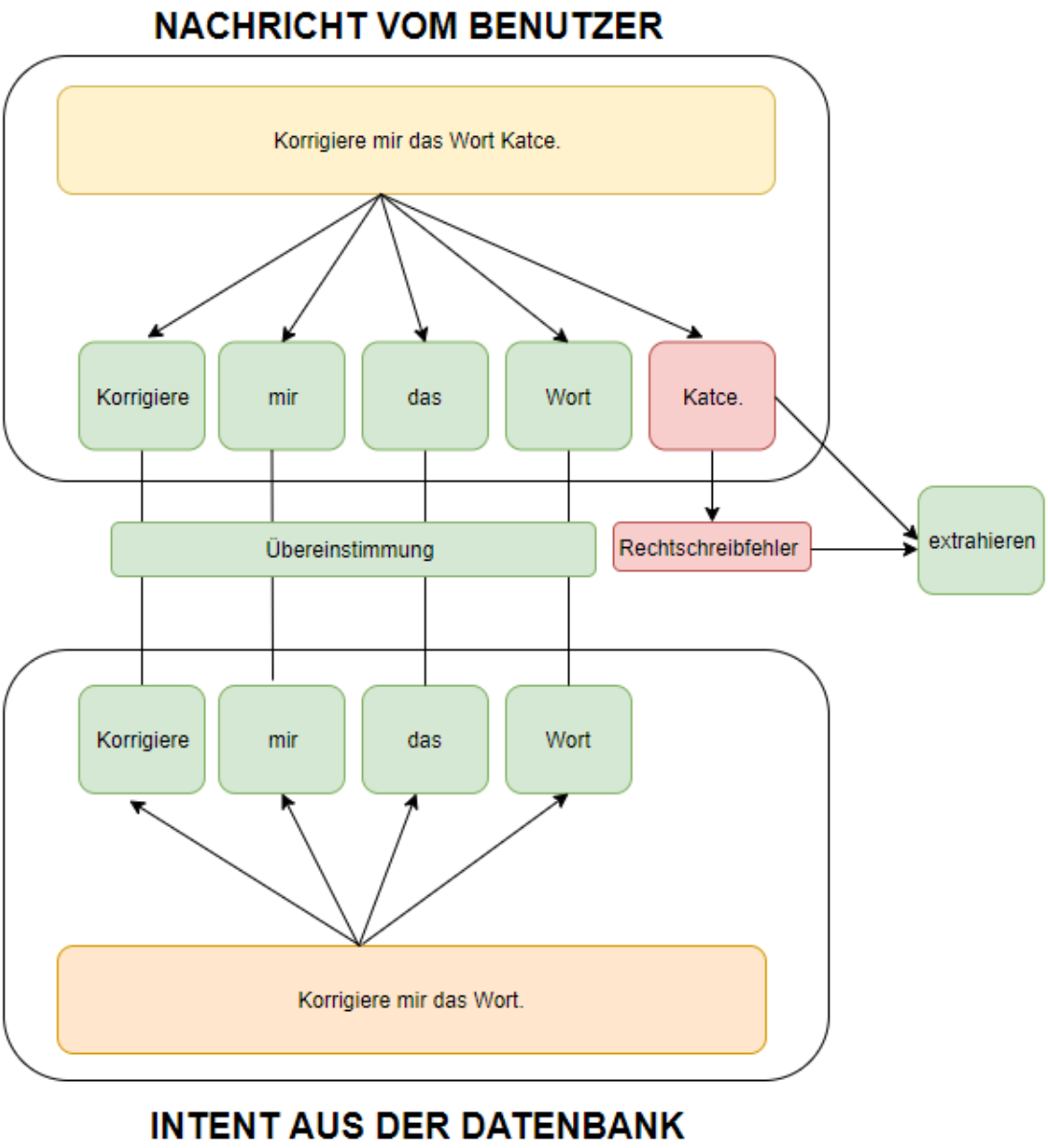

Abbildung 20: Extraktion Rechtschreibfehler/Entität

4.3.5 Korrekturvorschläge

Für die Korrektur der Rechtschreibfehler wurde das LanguageTool API verwendet. Zur Zeit der Entwicklung für den IDeRBlog-Chatbot wurde ein lokaler API-Server benutzt. Weitere Details der Serverbenutzung, wie Startbefehl, URL und Serverversion können in der folgenden Abbildung 21 entnommen werden.

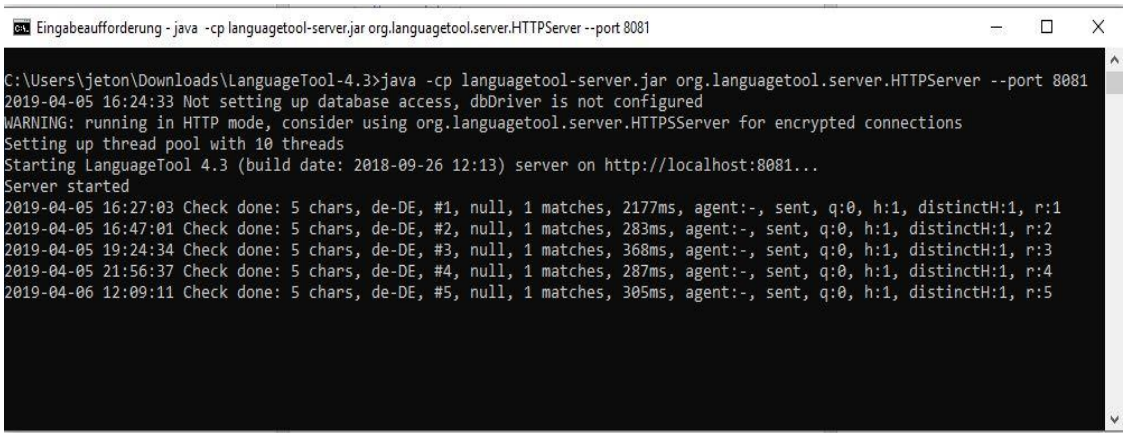

Abbildung 21: LanguageTool Server

Zum Erhalt der Korrektur für den Rechtschreibfehler für das extrahierte Wort „Katce" wurde wie folgend vorgegangen:

Das extrahierte Wort, welches einen Rechtschreibfehler enthält, wird zur Korrektur an den LanguageTool-Server API Endpunkt gesendet. Anschließend werden die Korrekturvorschläge im Backend mit den Testdaten (orthografische Begriffe), die in der Datenbank gespeichert sind, verglichen und der erste Vorschlag wird an die BenutzerInnen übermittelt. Sollten die BenutzerInnen den Korrekturvorschlag nicht bestätigen, wird aus den Testdaten der zweite Vorschlag, der eine höhere Übereinstimmung hat, übermittelt.

Die Übergabe der Schreibfehler an das LanguageTool-API Endpunkt erfolgt über den URI (aufrufbar auch im Browser), wie in der Auflistung 10 dargestellt wird.

```
http://localhost:8081/v2/check?text=Katce&language=de-DE
```

Auflistung 10: LanguageTool API Endpunkt

Die Übergabe der Schreibfehler und die gewählte Sprache sind Voraussetzungen, damit ein Korrekturvorschlag generiert wird. Der Aufruf erfolgt in der Methode getAllReplacements() im Backend und ist in der folgenden Auflistung 11 aufgelistet.

Anwendung im Prototypen

```
1 public static String getAllReplacements(String word) {
2     RestTemplate restTemplate = new RestTemplate();
3     String replacements =
restTemplate.getForObject("http://localhost:8081/v2/check?language=de
-DE&text="+ word.trim(), String.class);
4     return replacements;
5 }
```

Auflistung 11: Einsatz vom LanguageTool-Server für Korrekturvorschläge

Die Methode retourniert alle möglichen Korrekturvorschläge als JSON-Format am IDeRBlog-Chatbot-Backend. In dieser Liste werden ähnliche Wörter bzw. Korrekturvorschläge, die mit dem Rechtschreibfehler in Verbindung gebracht werden können, aufgelistet.

In der nachfolgenden Auflistung 12 wird die Liste der Korrekturvorschläge (replacements) für den Rechtschreibfehler „Katce" veranschaulicht aufgelistet.

```
{
  "software":{
    "name":"LanguageTool",
    "version":"4.3",
    "buildDate":"2018-09-26 12:13",
    "apiVersion":1,
    "premium":false,

    "status":""
  },
  "warnings":{
    "incompleteResults":false
  },
  "language":{
    "name":"German (Germany)",
    "code":"de-DE",
    "detectedLanguage":{
      "name":"English (US)",
      "code":"en-US"
    }
  },
  "matches":[
    {
      "message":"Möglicher Rechtschreibfehler gefunden",
      "shortMessage":"Rechtschreibfehler",
      "replacements":[
        {
          "value":"Katze"
        },
        {
          "value":"Sätze"
        },
        {
          "value":"Katzen"
        },
        {
          "value":"Katz"
        },
        {
          "value":"Käthe"
        },
        {
          "value":"Tatze"
        },
        {
          "value":"Kacke"
        },
        {
          "value":"Kitze"
        },
        {
          "value":"Käuze"
        },
        {
          "value":"Satze"
        },
        {
          "value":"Patze"
        },
        {
          "value":"Ratze"
```

```
062          },
063          {
064            "value":"Hatte"
065          },
066          {
067            "value":"Ganze"
068          },
069          {
070            "value":"Hätte"
071          },
072          {
073            "value":"Satz"
074          },
075          {
076            "value":"Kader"
077          },
078          {
079            "value":"Karte"
080          },
081          {
082            "value":"Marke"
083          },
084          {
085            "value":"Sitze"
086          }
087        ],
088        "offset":0,
089        "length":5,
090        "context":{
091          "text":"Katce",
092          "offset":0,
093          "length":5
094        },
095        "sentence":"Katce",
096        "type":{
097          "typeName":"UnknownWord"
098        },
099        "rule":{
100          "id":"GERMAN_SPELLER_RULE",
101          "description":"Möglicher Rechtschreibfehler",
102          "issueType":"misspelling",
103          "category":{
104            "id":"TYPOS",
105            "name":"Mögliche Tippfehler"
106          }
107        }
108      }
109    ]
110}
```


Auflistung 12: Korrekturvorschläge im JSON-Format

In dieser Auflistung 12 steht der Korrekturvorschlag für das Wort „Katce" gleich am Anfang der Liste und wird an die BenutzerInnen als erstes vorgeschlagen. Dies ist das „Bestcase-Szenario".

Das "Worstcase-Szenario" könnte auftreten, wenn die BenutzerInnen in Umgangssprache oder in unterschiedlichen Sprachvariationen schreibt. Ein typischer Rechtschreibfehler wäre „Korrigiere mir das Wort **Vata**".

In diesem Bespiel kann das LanguageTool-API nicht die gewünschte Ergebnisse liefern, da solche orthografischen Fehler bzw. Sprachvariationen vom LanguageTool-API schwer bzw. gar nicht erkannt werden können.

Mit dem Wort „Vata" ist hier „Vater", der Elternteil gemeint und nicht das Wort „Fatal", welches als erster Korrekturvorschlag ganz oben in der Liste aufgelistet ist (siehe Auflistung 13). Eine Konversation mit den BenutzerInnen zu führen, bis das korrekte Wort von den BenutzerInnen angenommen wird, macht hier keinen Sinn. Es müssten neun weitere Korrekturvorschläge den BenutzerInnen vorgeschlagen werden. Dies würde die BenutzerInnen, bis der korrekte Vorschlag geliefert wird, ungeduldig machen bzw. besteht die Möglichkeit, dass die BenutzerInnen die Konversation aufgibt und beendet.

In der anschließenden Auflistung 13 werden die Korrekturvorschläge für das Wort „Vata" veranschaulicht.

```
"matches": [
    {
      "message": "Möglicher Rechtschreibfehler gefunden",
      "shortMessage": "Rechtschreibfehler",
      "replacements": [
        {
          "value": "Fatal"
        },
        {
          "value": "Dada"
        },
        {
          "value": "Fade"
        },
        {
          "value": "Data"
        },
        {
          "value": "Fata"
        },
        {
          "value": "Vati"
        },
        {
          "value": "Veda"
        },
        {
          "value": "Vita"
        },
        {
          "value": "Fad"
        },
        {
          "value": "Hat"
        },
        {
          "value": "Vater"
        },
        {
          "value": "Fand"
        },
        {
          "value": "Bad"
        },
    }
]
```

Auflistung 13: Worst Case Szenario

Um diese Problem zu lösen, musste die Datenbank mit Testdaten bzw. mit orthografischen Begriffen gespeist werden. Diese umgangssprachlichen Begriffe und die gesuchten Wörter müssen in einer Beziehung mit stehen. Für diese Beziehung wird ein Häufigkeitsfaktor definiert und berechnet, um einen genauen Korrekturvorschlag zu liefern. Dieser Häufigkeitsfaktor muss

ständig aktualisiert werden, wenn die BenutzerInnen im Zuge der Konversation diesen Korrekturvorschlag entgegennehmen. Damit wird gewährleistet, dass die Korrekturvorschläge kontinuierlich exakter werden.

Für den Begriff „Vater" werden z.B. im Raum Steiermark, die umgangssprachlichen Begriffe wie „Vata" und „Vota" verwendet.

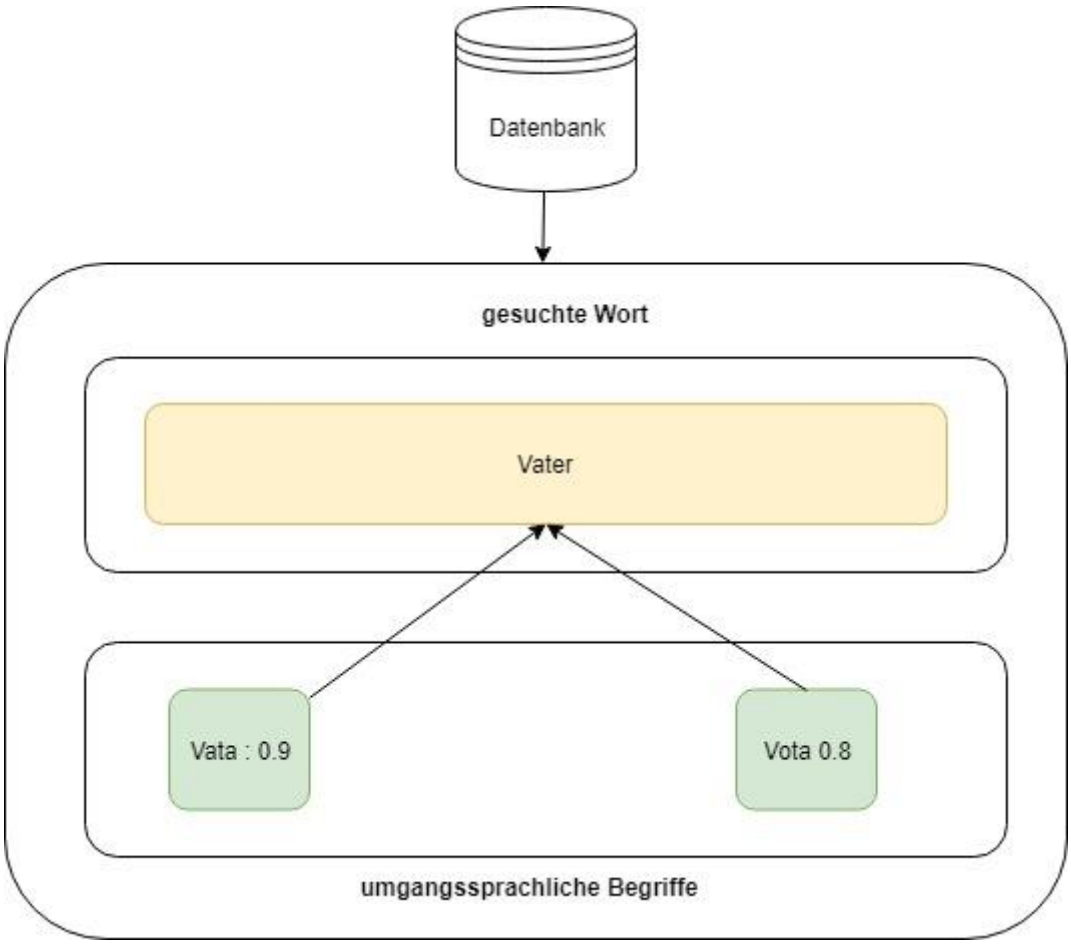

Abbildung 22: Häufigkeitsfaktor

Die gesuchten umgangssprachlichen Begriffe, die einen höheren Häufigkeitsfaktor aufweisen, werden zur Korrektur entnommen.

4.3.6 Generieren einer Antwort

Die Generierung und der Versand der Nachricht vom Webbackend bis zum Webfrontend wird in diesem Abschnitt beschrieben und grafisch dargestellt. In der Abbildung 233 werden alle Schritte noch einmal bildlich dargestellt.

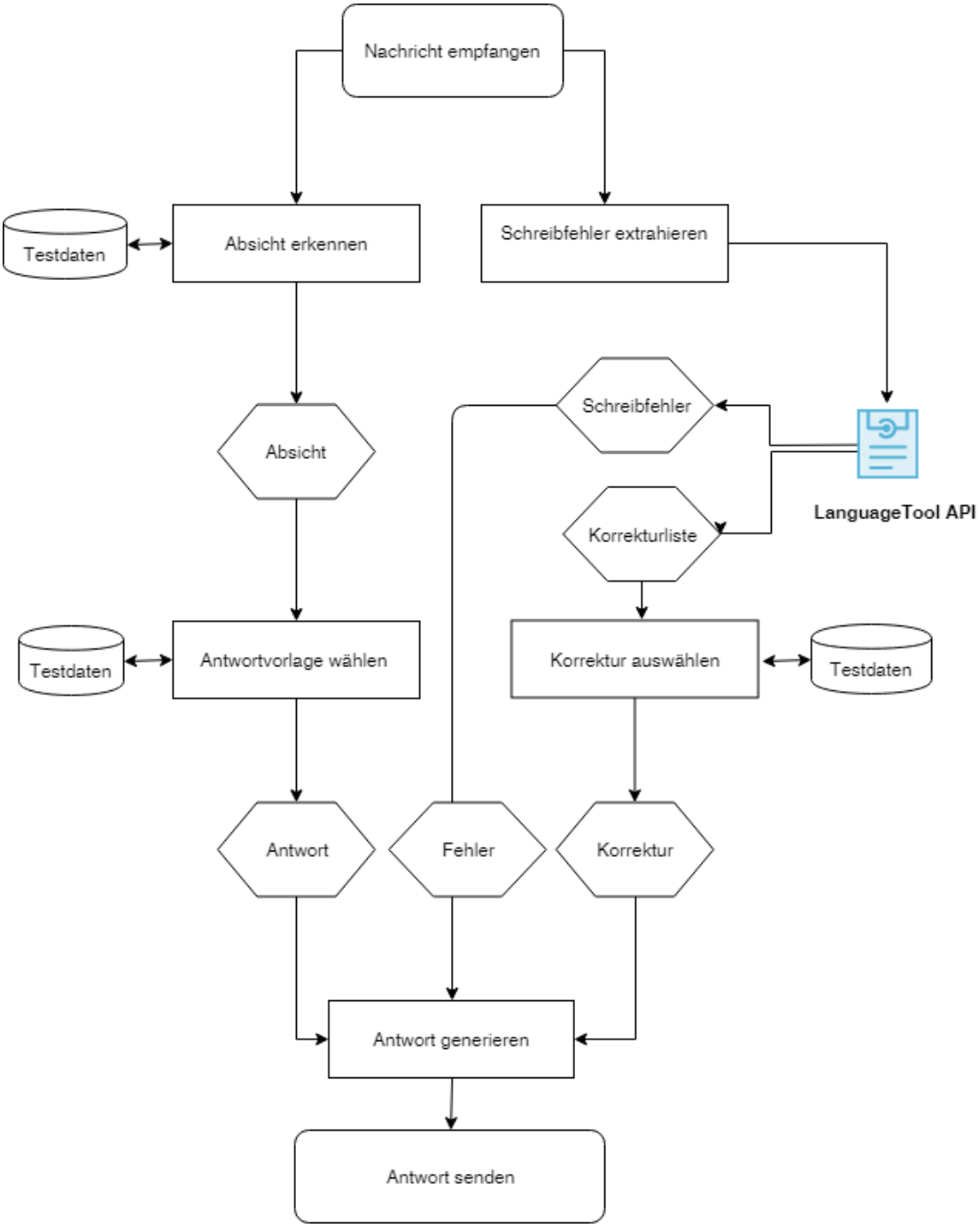

Abbildung 23: Ablaufdiagramm von der Nachricht bis zur Antwort

Nachdem die Absicht (Intent) des Benutzers erkannt wurde, wird automatisch auch eine Antwortvorlage von den Testdaten aus der Datenbank ausgewählt. Die Antwortvorlage enthält die Textnachricht (response) und die nötigen drei Slots, die als Platzhalter für den Schreibfehler und die Korrektur dienen. Für die Absicht „Schreibfehler" (MisSpellingIntent) wurde aus der Datenbank die Antwortvorlage „Das Wort **{Schreibfehler}** ist mir nicht bekannt, meintest du vielleicht **{Korrektur}**, **{Ergänzung}?"**

Als nächstes müssen nur Schreibfehler und die Korrektur jeweils an der richtigen Stelle (Slot) einfließen, um die Antwort zu generieren. Sollte für die Korrektur zusätzlich noch eine Ergänzung nötig sein, muss diese zusätzlich in die Antwort einfließen (siehe Abbildung 244)

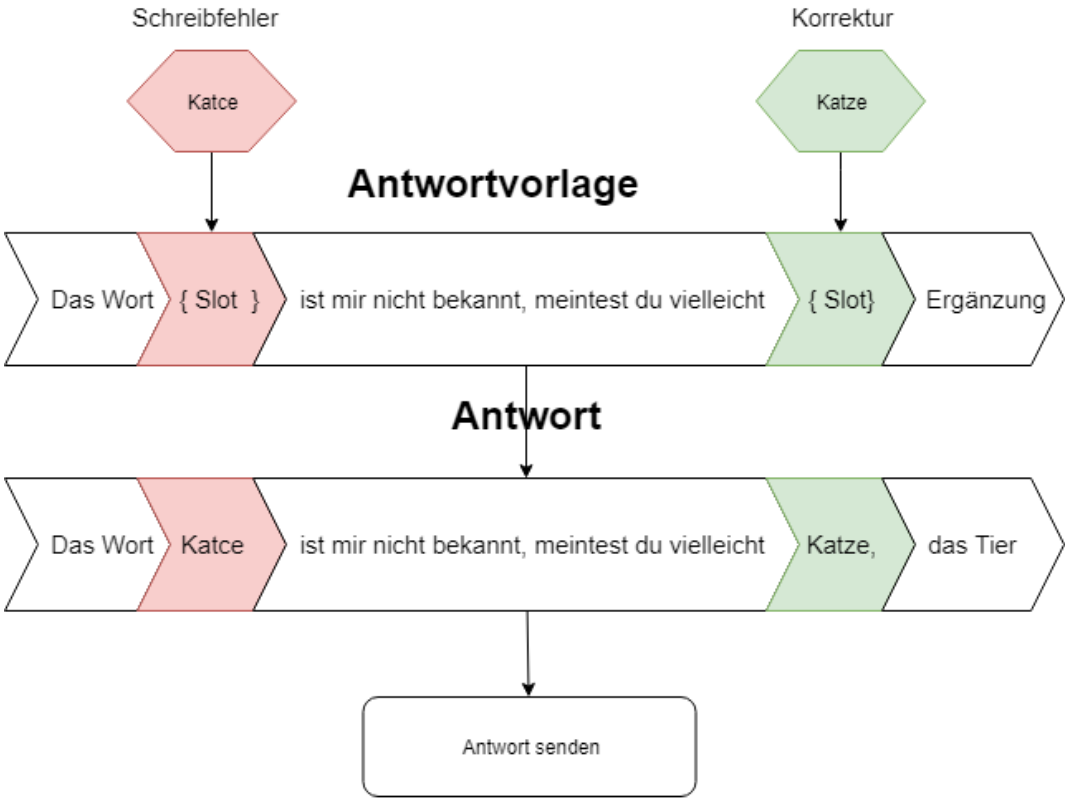

Abbildung 24: Generierung der Antwort

Zur Demonstration wurde hier das Beispiel aus dem „Bestcase-Szenario" übernommen.

5 TESTUNG UND AUSBLICK

In diesem Kapitel wird im Abschnitt 5.1 die Testung und deren Ergebnisse beschrieben und im Ausblick im Abschnitt 5.2 werden weitere Überlegungen für den IDeRBlog-Chatbot angeführt.

5.1 Testung

Abschließend wurde eine Testung des Prototypen bzw. des IDeRBlog-Chatbots durchgeführt. Es wurden eine Begrüßung, Smalltalk, Fragestellungen zu den Rechtschreibfehlern und Fragestellungen zu orthographischen Fehlern/Begriffen geprüft.

Testung 1: Begrüßung und Smalltalk

Fragestellung an den IDeRBlog-Chatbot:

- „Hallo!"
- „Wie heißt du?"
- „Wie geht es dir?"

Es wurden alle Absichten aus den Nachrichten erkannt und es wurden korrekte Begrüßungen für die Testperson generiert.

Testung 2: Fragestellungen zu den Rechtschreibfehlern

Fragestellung an den IDeRBlog-Chatbot:

- „Wie viele Fehler habe ich meinem Text?"
- „Schreibt man Appril mit zwei p?"
- „Habe ich Walt richtig geschrieben?"

Auch bei der Testungen der Rechtschreibfehler wurden alle Absichten aus den Nachrichten erkannt. Es wurde die richtige Anzahl der Fehler erkannt und alle Rechtschreibfehler wurden die richtigen Korrekturvorschläge geliefert.

Testung 3: Fragestellungen zu orthographischen Fehlern/Begriffen

Fragestellung an den IDeRBlog-Chatbot:

- „Korrigiere mir das Wort **Vata**" -> Vater
- „Korrigiere mir das Wort **Ai**" -> Ei

Beim Testdurchlauf schlug der Korrekturvorschlag vom LanguageTool-API fehl, weil die Datenbank nicht mit orthografischen Fehlern/Begriffen gefüllt war. Nachdem in die Datenbank orthografische Fehler hinzugefügt wurden, wurde schon beim zweiten Mal der korrekte Korrekturvorschlag ausgegeben. Durch die Aktualisierung des Häufigkeitsfaktors, der bei der Annahme des Korrekturvorschlags von den BenutzerInnen gemacht wird, wurden die Korrekturvorschläge immer genauer.

Testung 4: Textüberprüfung mit grammatikalischen Fehlern (Artikel)

Fragestellung an den IDeRBlog-Chatbot:

- „Sagt man die Katze oder der Katze?"

Bei dieser Testung wurde eine grammatikalische Fragestellung vom Verfasser gemacht und die Absicht wurde erkannt, aber die Korrektur schlug fehl, da die Korrektur von grammatikalischen Fehlern im Rahmen dieser Masterarbeit bzw. bei der Aufgabenstellung nicht gefordert war.

5.2 Ausblick

Um den IDeRBlog-Chatbot schlauer zu machen, müsste der IDeRBlog-Chatbot mit einer größeren Datenmenge an orthografischen Begriffen und Sprachvariationen erweitert werden. Um die Genauigkeit zu steigern, sollte der IDeRBlog-Chatbot auch häufig genutzt werden. Grammatikalische Fehler wurden bei der Testung nicht erkannt, weil dafür keine Intents definiert wurden. Wie gut ein Chatbot Antworten generiert, hängt von der Menge der Intents in seiner Datenbank und seiner Lernfähigkeit ab.

Die Gestaltung des IDeRBlog-Chatbots sollte kindergerecht erfolgen und eine bedienungsfreundliche Oberfläche mit ansprechenden Farben und Symbolen aufweisen, um die Motivation den BenutzerInnen zu steigern. Zusätzlich könnte der IDeRBlog-Chatbots mit einem ansprechenden Avatar ausgestattet sein und es sollte die Möglichkeit einer Schriftgrößeneinstellung und eventuell auch eine Sprachausgabe für die Barrierefreiheit des IDeRBlog-Chatbots geboten werden.

6 ZUSAMMENFASSUNG

Ziel dieser Masterarbeit war, einen Chatbot für den IDeRBot-Lernplattform der TU-Graz zu entwickeln und diesen in der Lernplattform zu integrieren. Der Chatbot soll wie ein Tutor bei der Rechtschreibüberprüfung fungieren und mit den SchülerInnen kommunizieren. Durch den Einsatz bzw. die Verwendung des Chatbots sollen die NutzerInnen einen Mehrwert beim Erlernen der deutschen Sprache erhalten. Zusätzlich soll die Interaktion den NutzerInnen anregen, mehr Texte in der deutschen Sprache zu verfassen.

Zuerst wurde mit Heranziehen der Literatur ein Überblick über Chatbots im Allgemeinen gegeben. Er wurde der Begriff Chatbot definiert und Tests, mit denen Qualitätsstandards ermittelt werden, aufgezeigt. Thematisiert wurde ebenso die historische Entwicklung und die Funktionsweisen von Chatbots. Einsatzgebiete, in welchen Bereichen Chatbots bereits erfolgreich eingesetzt werden, wurden ebenso aufgezeigt. Im Anschluss wurde auf den IDeRBlog-Chatbots eingegangen, indem eingesetzte Technologien, Entwicklungsumgebungen und Bibliotheken beschrieben wurden. Es wurde HTML für die Struktur des IDeRBlog-Chatbots-Frontends verwendet und das Layout wurde mit CSS definiert. Für die Kommunikation zwischen Webfrontend und Webbackend wurde JavaScript, unter Verwendung von WebSockets, eingesetzt. Das Webbackend wurde mit Java und Java-Spring-Boot entwickelt. Im Anschluss wurden kurz die eingesetzten Bibliotheken erklärt. Ein Hauptaugenmerk wurde dabei auf das LanguageTool gelegt, da dieses Korrekturvorschläge für den IDeRBlog-Chatbot liefert. Im Kapitel vier, welches sich mit der Umsetzung des Prototypen beschäftigt wurde

detaillierter auf den Einsatz der bereits genannten Technologien im Prototypen eingegangen. Beim Design auf der Clientseite im Webfrontend wurde

Unter Berücksichtigung der Aufgabestellung wurde im Webfronten des IDeRBlog-Chatbots eine eigene Lösung implementiert. Die Bedienoberfläche wurde entworfen und an den bestehenden IDeRBlog- Lösung angepasst, damit die Integration problemlos erfolgen kann. Im Webbackend wurde für die Absichten Standardabsichten definiert, um ein besseres Verständnis der natürliche Sprache an die BenutzerInnen zu vermitteln. Es musste auch eine Strategie überlegt werden, wie man orthographische Fehler und Sprachvariationen erkennt und korrigiert.

Nachdem eine Testung durchgeführt wurde, kann gesagt werden, dass der IDeRBlog-Chatbot für die definierten Absichten gute Ergebnisse lieferte. Die Fehlerkorrektur für die vorhandene orthografischen Schreibfehler war zufriedenstellend. Es stellte sich heraus, dass die Fehlererkennungsrate stark an den vorhandenen Testdaten abhängt. Somit ist eine Erweiterung der Datenbank mit Testdaten erforderlich. Eine weitere Optimierung des IDeRBlog-Chatbots wäre die Fehlererkennung von grammatikalischen Fehlern.

Zusammenfassend ist anzumerken, dass die korrekte Generierung von Antworten durch einen Chatbot von der Menge der Intents in seiner Datenbank und Lernfähigkeit abhängt.

LITERATURVERZEICHNIS

Ackermann Philip (2018): JavaScript. Das umfassende Handbuch. 2. aktualisierte und erweiterte Auflage. Bonn: Rheinwerk Verlag.

Ahmad, Nahdatul Akma; Hamid, Mohamad Hafiz Che; Zainal, Azaliza; Rauf, Muhamad Fairuz Abd; Adnan, Zuraidy (2018): Review of Chatbots Design Tecniques. In: International Journal of Computer Applications (0975-8887), Volume 181, No. 8.

Braun, Alexander (2003): Chatbots in der Kundenkommunikation. Springer-Verlag Berlin Heidelberg.

Cornelißen Patrick/Piefel Michael/Sparkowsky Alexander (2018): Spring Boot 2 Fundamentals: Build and deploy production-ready microservices within the Java ecosystem. Packt Publishing Ltd.

Crockford Douglas/Klicman Peter (2008): Das Beste an JavaScript. O`Reilly Verlag GmbH & Co. KG.

DATACOM Buchverlag GmbH (2019): IDE (integrated development environment). In: https://www.itwissen.info/IDE-integrated-development-environment-Integrierte-Entwicklungsumgebung.html (besucht am 25.03.2019).

Deshpande, Aditya; Shahane, Alisha; Gadre, Darshana; Deshpande Mrunmayi; Prod. Dr. Joshi, Prachi M (2017): A Survey of Various Chatbot Implementation Techniques. International Journal of Computer Engineering and Applications, Volume XI. In: https://pdfs.semanticscholar.org/8e60/5c49d4a7cba9bf077d97b401ba78aafe693f.pdf (besucht am 27.11.2018).

Flanagan David (2012): JavaScript kurz & gut. 4. Auflage. Köln: O`Reilly Verlag GmbH & Co.KG.

Futuregram (2018): Chatbots – Künstliche Intelligenz im Messenger? In: https://futuregram.trendone.com/chatbots/ (besucht am 27.11.2018).

1&1 IONOS SE (2019): Spring – das Framework für komplexe Java-Applikationen. In: https://www.ionos.de/digitalguide/websites/web-entwicklung/spring-framework-das-steckt-in-dem-java-grundgeruest/ (besucht am 25.03.2019).

Keller Daniela (2018): Eine Einführung in das Languagetool. In: https://projects.fzai.h-da.de/linguistic-complexity/wp-content/uploads/2018/08/Einf%C3%BChrung-in-das-LanguageTool_DanielaKeller.pdf (besucht am 25.03.2019).

Kimberlin Michael (2015): Reducing Boilerplate Code with Project Lombok. In: http://jnb.ociweb.com/jnb/jnbJan2010.html (besucht am 25.03.2019).

Laborenz Kai (2015): CSS. Das umfassende Handbuch. 3 Auflage. Bonn: Rheinwerk Verlag.

LanguageTooler GmbH (2019): Funktionen und Hilfe. In: https://languagetool.org/de/ (besucht am 25.03.2019).

Lisbach Bertrand (2011): Linguistisches Identity Matching. Paradigmenwechsel in der Suche und im Abgleich von Personendaten. 1. Auflage. Germany: Springer Fachmedien Wiesbaden GmbH.

Mathur Amit (2002): How to build Eliza Chatterbot. In: http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=5369&lngWId=3 (besucht am 27.11.2018).

Mobile app design and development company (2019): 6 Ways to Create AI for Your Chatbot: Ready-Made Solutions and Alternatives. In:

https://tecsynt.com/blog/6-ways-to-create-ai-for-your-chatbot (besucht am 27.11.2018).

Möbus, Claus; Eißner, Andreas; Feindt, Jan; Janser, Claudia; Krefeldt, Jens; Sieverding, Sven; Sölbrandt, Stefan; Stumpe, Jörg; de Vries, Holger; Willer, Stefan (2006): Web-Kommunikation mit OpenSource. Chatbots, Virtuelle Messen, Rich-Media-Content. Springer-Verlag Berlin Heidelberg.

Onlim GmbH (2018): 7 große Marken die bereits Chatbots einsetzen. In: https://onlim.com/7-grosse-marken-die-bereits-chatbots-einsetzen/ (besucht am 27.11.2018).

Schwichtenberg Holger (2019): Was ist Websockets? In: https://www.it-visions.de/glossar/alle/6283/Websockets.aspx (25.03.2019).

Simons, Michael (2018): Spring Boot 2. Moderne Softwareentwicklung mit Spring 5. 1. Auflage. Heidelberg: dpunkt.verlag GmbH.

Schlündt, Thomas (2018): Wo werden Chatbot derzeit eingesetzt? In: https://ebusiness2020.wordpress.com/2018/06/12/wo-werden-chatbots-derzeit-eingesetzt/ (besucht am 30.11.2018).

Spring by Pivotal (2019): WebSocket Support. In: https://docs.spring.io (besucht am 05.04.2019).

Storp, Michaela (2002): Chatbots. Möglichkeiten und Grenzen der maschinellen Verarbeitung natürlicher Sprache. In: https://www.mediensprache.net/networx/networx-25.pdf (besucht am 28.11.2018).

Techopedia (2019): IntelliJ IDEA. In: https://www.techopedia.com/definition/7755/intellij-idea (besucht am 25.03.2019).

The Apache Software Foundation (2019): Introduction to the POM. In: https://maven.apache.org/guides/introduction/introduction-to-the-pom.html (besucht am 25.03.2019).

The Project Lombok Authors (2009-2019): Project Lombok. In: https://projectlombok.org/ (besucht am 25.03.2019).

W3C (2019): HTML 5.2. W3C Recommendation. Introduction. In: https://www.w3.org/TR/html5/introduction.html#introduction (zuletzt besucht am 1.4.2019).

Weigel Andreas (2019): Was ist Bootstrap? In: https://www.bootstrapworld.de/was-ist-bootstrap.html (besucht am 30.03.2019).

Weßendorf Matthias (2011): WebSocket: Annäherung an Echtzeit im Web. In: https://www.heise.de/developer/artikel/WebSocket-Annaeherung-an-Echtzeit-im-Web-1260189.html?seite=all (besucht am 25.03.2019).

Willmes, Anna (2014): Chatbots: Was ist das und wie funktionieren sie? In: https://famiville.wordpress.com/2014/11/24/chatbots-definition-funktionsweise-einsatzgebiete-starken-schwachen/#more-139 (besucht am 27.11.2018).