Design eines interaktiven Agents zur Unterstützung von Studierenden

Bachelorarbeit

von

Lukas Miliunas

aus

Vilnius, Litauen

vorgelegt am

Lehrstuhl für Rechnernetze
Prof. Dr. Martin Mauve
Heinrich-Heine-Universität Düsseldorf

Oktober 2018

Betreuer:
Christian Meter, M. Sc.

Zusammenfassung

Das Ziel dieser Arbeit ist es einen Chatbot für Informatikstudenten zu erzeugen. In erster Linie sollte es für die Studenten des ersten Semesters geeignet sein. Am Anfang des Studiums stellt sich jeder Student fast genau die selben Fragen und ist ziemlich verwirrt. Mehrere Mitarbeiter an der Heinrich–Heine–Universität bekommen eine Vielzahl gleicher Fragen. Meistens genügt es die Studenten auf passende Internet–Seiten zu verweisen oder es einfach in ein paar kurzen Sätzen zu beantworten. Da sich Fragen sowie Antworten ständig wiederholen, könnte ich durch den Chatbot sowohl künftige Studierende glücklich machen, als auch sehr viel Zeit für die Mitarbeiter unserer Universität sparen. Ein weiteres Ziel – der Chatbot soll ein Begleiter am Anfang des Studiums sein. Jeder Studierende sollte sich am Anfang sicher fühlen – ein Chatbot kann das schaffen. Sicher fühlt man sich genau dann, wenn man alles weiß, was man wissen soll. Dies ist momentan nicht der Fall. Es gibt viele Studenten, welche anfangs einige Sachen nachfragen, diese nicht aufschreiben und dann sofort wieder vergessen. Manche sind zu schüchtern um nochmal zu fragen. Ein Chatbot antwortet immer, sogar auf die gleichen Fragen tausendmal hintereinander. Diese Bachelorarbeit erfasst die Entwicklung des Chatbots und Ermittlung der richtigen Fragen beziehungsweise Antworten. Der implementierte Bot sollte sich in folgenden und noch weiteren Themen auskennen: Stundenplan, Lageplan, Ilias, SSC, Bachelorarbeit, Fachschaft, alle Lehrstühle der Informatik, Credit Points, Studium Universale, Internet, BAföG, Mensa, Nachhilfe, Wohnheim, Veranstaltungen, Krankheit, Klausurwiederholung, Klausuranmeldung, Nebenfach, Anerkennung, Dekanat, Sekretariat, Bücher, Laptop, Software, Übungsgruppen, Klausurzulassung, Termine, Fristen, AUAS, ZIM.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Motivation

An Universitäten ist die Zahl fachlicher und organisatorischer Fragen von Studierenden an die Mitarbeiter der Servicestellen und der Lehrstühle, beispielsweise in Form von E–Mails, sehr hoch. Dies erzeugt für die Angestellten der Universität einen hohen Arbeits– und Zeitaufwand. Doch im Endeffekt beantworten die Mitarbeiter immer wieder ähnliche oder gleiche Fragen, woraus eine enorme Ineffizienz entsteht. Mithilfe der Problemstellung dieser Arbeit sowie durch eine kurze Forschung ist ersichtlich geworden, dass es für dieses Problem eine Lösung gibt. Ziel dieser Arbeit ist es, eine Art Kundenservice für die Studierenden zu generieren, welcher ohne weitere Kosten für die Universität verfügbar wäre. Um dies zu erreichen, wird im Folgenden ein Chatbot entwickelt, der auf die häufigsten Fragen der Studierenden schnell und unkompliziert antworten kann. Zur Umsetzung dieses Projektes können beliebig viele Frameworks, Hilfen und Anwendungen benutzt werden. Zunächst erfolgt eine Benennung aller möglichen Frameworks, um letztlich auf die beste Lösung zu kommen. Anschließend werden die ausgewählten Instrumente vollständig automatisiert und als Endprodukt wird ein interaktiver Chatbot entstehen. Erstrebenswert bei der Entwicklung ist, dass dieser Bot nur einen minimalen Aufwand an Wartung erfordert und höchstpräzise funktionieren wird. Es soll sich am Ende für die Nutzer anfühlen, als ob sie sich mit einem menschlichen Kundenservice unterhalten würden, obwohl Sie im Wirklichkeit mit einem Computer, einem Bot, kommunizieren.

Kapitel 2

Recherche

Um eine möglichst effiziente und verständliche Nutzung des Bots zu ermöglichen, ist es zunächst notwendig, herauszufinden und darzustellen, welche Werkzeuge zur Entwicklung und zur Umsetzung benötigt werden. Zum Einen wird eine Schnittstelle gebraucht, damit es möglich ist mit dem Bot zu kommunizieren. Zum Anderen wird ein Framework („Dialogflow“, „microsoft bot.framework“, „wit.ai“ und einige weitere private Entwicklungen) notwendig sein. Des Weiteren stellt Logik einen elementaren Faktor dar, denn durch diese wird alles vernetzt und zum Laufen gebracht (z.B. auf „Java“ oder „Python“ basierte Skripte). Ebenso wird auch ein Server benötigt, auf dem der Bot „leben“ wird. Im Folgenden wird erläutert, welche Mittel gewählt werden. Als Erstes erfolgt die Auswahl eines Frameworks, welches den Bot technisch beschreibt.

Es ist nicht ganz einfach sofort zu bestimmen, welches Framework das geeignetste für einen Studierendenbot wäre, da es zur Zeit mehr als 20 Bots [Dat18] gibt, die zur Auswahl stehen. Ein wichtiges Auswahlkriterium ist, wie lange ein Framework bereits auf dem Markt ist. IBM Watson, Microsoft Bot Framework, Dialogflow, ChatScript, wit.ai und Pandorabots sind alle schon mehr als 10 Jahre [Dat18] aktiv und haben dadurch sehr viel Erfahrung erlangt(API.AI ist zwischendurch zu Dialogflow umbenannt worden). Insbesondere bei einem Bot ist es wichtig, dass die Entwickler bereits alle möglichen Arten von Fehlern erfahren haben. Dies hilft bei der Weiterentwicklung und ermöglicht es, diese Fehler zu korrigieren und somit die Fehlerquote zu minimieren. Daher haben die älteren Bots einen deutlichen Erfahrungsvorsprung. Jedoch gibt es auch zwischen den neueren Bots einen starken Konkurrenten. Wit.ai [1] wurde erst im Jahr 2013 gegründet, hat aber schon über 160.000 aktive Benutzer.

[1] `https://wit.ai/`

Des Weiteren ist ein wichtiges Auswahlkriterium, dass das Framework Stärken im Bereich „Chatbot“ ausweist. Aufgrund weiterer Recherche im Internet [Har18] und ergänzender Gespräche mit einigen Mitstudierenden, die Bot Frameworks schon ausprobiert haben, wurde die Entscheidung getroffen, die Auswahl auf die folgenden Frameworks zu begrenzen: Microsoft Bot Framework, Dialogflow und wit.ai. Microsoft Bot Framework hat eine Limitation [Mic18a]. Bei diesem sind 10.000 Nachrichten pro Monat kostenlos. Für weitere 1000 Nachrichten ist eine Gebühr von 42 Eurocents zu entrichten.

	Dialogflow	wit.ai	Microsoft Bot Framework
Owner	Google	facebook	Microsoft
Slot–filling	ja	nein	nein
Price	free	free	only 10,000 messages free
Pre–Build Entities	yes	yes	yes

Tabelle 2.1: Vergleich möglicher Bot-Frameworks

Eine weitere Recherche hat ergeben, dass wit.ai kein „slot–filling“ unterstützt [Jac18]. Dies bedeutet, dass mit Dialogflow durch „slot–filling“ [Dia18g] bestimmte Parameter als „unbedingt benötigt“ bezeichnet werden können. Ohne diese Parameter wird Dialogflow nicht weiter vorgehen. Bei wit.ai hingegen ist es notwendig, das ganze Gespräch auszuwerten, um die noch nicht vorhandenen unbedingt notwendigen Parameter festzustellen. Erst nach der gesamten Auswertung ist eine weitere Anfrage möglich. Wit.ai ist also auch eine mögliche Alternative, die akzeptabel wäre. Dennoch stellt Dialogflow eine viel effizientere (und offensichtlich schnellere) Lösung dar, weil die unbedingt notwendigen Parameter direkt nachgefragt werden.

Theoretisch wäre es möglich, sowohl wit.ai als auch Dialogflow zu benutzen, da die Grundvoraussetzungen für einen funktionsfähigen Bot bei beiden erfüllt sind. Aber aufgrund der eben genannten Vorteile und persönlicher Präferenzen wird in dieser Arbeit Dialogflow gewählt. Dieses Framework definiert den Bot verständlich und benutzerfreundlich.

Die gewählte Gesprächsstruktur entspricht dem natürlichen und realen Gespräch unter Menschen und ist dadurch leichter nachvollziehbar. Dialogflow imitiert ein echtes Gespräch mit einer Person, sammelt dadurch bestimmte Daten, bündelt diese in einem Webhook und schickt dieses an das sogenannte „back–end“ weiter. Damit alle von Dialogflow übermittelten Da-

ten, die bestimmter Handlungen bedürfen, verarbeitet und bestimmte Handlungen ausgelöst werden können, ist es notwendig, ein Programm zu entwickeln, das eine solche Verarbeitung ermöglicht. Als Programmiersprache wird Python gewählt. Ein wichtiger Grund für diese Wahl ist, dass Dialogflow einen API [Goo18] für Python bereitstellt.

Das Programm wird als ein Server funktionieren, der eine Laufzeitumgebung dafür braucht. Eine primitive Lösung wäre, einfach einen Laptop zu Hause stehen zu lassen, der ununterbrochen für Dialogflow bereit stehen und aktiv einen Internetanschluss benutzen würde. Dies wäre jedoch ein zu hoher Aufwand für eine Arbeit, die durch eine Lösung von Cloud [Mic18b] viel effizienter sein könnte. Zur Auswahl stehen dabei Public und Private Cloud. Da Private Cloud von kleinen privaten Unternehmen angeboten wird, kostet es auch viel Geld. Daher ist Public Cloud für unsere Nutzung besser geeignet, denn bei diesem werden bei dem geringen Ressourcenverbrauch des Bots entweder gar keine oder nur sehr geringe Kosten anfallen.
Die größten Public Cloud Platforms [NEW18] sind Amazon Web Services, Microsoft Azure und Google Cloud Platform. Hier ist Amazon seit einigen Jahren [Jor18] eindeutig der führende Anbieter und hat zusätzlich auch die beste Erreichbarkeit [AND18].

Die Ressourcen, die für den Bot verwendet werden, sind so klein, dass sie kostenlos sind, da sie den Rahmen von Free–Trier [Ama18f] des AWS nicht gänzlich ausfüllen. Free–Trier ist die AWS Kapazität, die im Rahmen eines kostenlosen AWS–Kontos entgeltfrei benutzt werden darf. Aufgrund der oben genannten Vorteile wird Amazon Web Services als Cloudlösung für den Dialogflow–Chatbot gewählt.

Die Instrumente für die Erstellung eines Bots sind nun ausgewählt. Als nächstes ist es besonders wichtig, mehrere Dokumentationen und Implementierungssbeispiele zu betrachten, um den bestmöglichen Weg für die Implementierung herauszufinden.

Kapitel 3

Erste Gedanken

Um das in der Einleitung beschriebene Problem der vielen Fragen der Studierenden zu lösen, muss das Problem zunächst genau verstanden werden. Erst danach ist es möglich, auf die Einzelheiten einzugehen. Dabei habe ich zunächst alle Probleme betrachtet, die ich am Anfang meines Studiums hatte. Zu Beginn war nicht bekannt, wie das Universitätsleben funktioniert. Bei jeder Frage wurde zunächst nach Antworten auf der Universitätsinternetseite gesucht. Falls dadurch kein Erfolg erzielt werden konnte, habe ich mich an die Mitarbeiter der Universität Düsseldorf gewandt. Sie waren immer hilfreich, aber genau diese Vorgehensweise soll durch meine Bachelorarbeit effizienter hinsichtlich des Zeit–, Arbeits– und Kostenaufwandes gestaltet werden. Doch nicht nur meine Fragen sind wichtig, sondern auch die von den anderen Studierenden. Daher wurden viele Mitstudierende gefragt, welche Probleme sie am Anfang des Studiums hatten und wie sie diese Probleme lösen konnten. Durch meine Joberfahrung als Tutor sowohl für Erstsemester als auch in Vorkursen habe ich selber sehr viele Fragen von Studierenden erhalten und kann dadurch das Problem noch besser verstehen. Basierend auf all diesen Daten wird im Folgenden ein Fragenkatalog entworfen.

Themen:

- Neu an der Uni:
  - Fragen:
    * Wie kann ich mich mit dem Internet verbinden?
    * Bietet die Heinrich Heine Universität WLAN an?

        * Welche Vorlesungen habe ich?
        * Welche Übungen habe ich?
        * Wer kann mir helfen, meinen Stundenplan zusammenzustellen?
        * Wann fangen welche Vorlesungen an?
        * Was ist AUAS?
        * Was ist ILIAS?
        * Wo finde ich einen Kiosk?
- Klausuren:
    - Fragen:
        * Wo muss ich mich für die Klausuren anmelden und wann?
        * Wann sind die Klausuren?
        * Was mache ich, wenn ich krank bin?
        * Welche Voraussetzungen muss ich erfüllen, um die Zulassung zu bekommen?
- Organisatorisches:
    - Fragen:
        * Wo kann ich Rechnernetze/ Datenbanken/ Bioinformatik/ „andere Lehrstühle der Informatik“ finden?
        * Wie finde ich das Sekretariat?
        * Wie melde ich meine Bachelor–/Masterarbeit an?
        * Wo ist die Bibliothek?

- Freizeit:
  - Fragen:
    - Gibt es Sport–/Theater–/Tanz–Kurse?
    - Wo finde ich aktuelle Veranstaltungen der Universität Düsseldorf?
- Wohnen:
  - Fragen:
    - Gibt es Studentenwohnheime?
    - Wer kann mir bei der Wohnungssuche helfen?
    - Wie lange kann man im Studentenwohnheim wohnen?
    - Gibt es Fristen für die Bewerbung für die Wohnheime?
- Nachhilfe:
  - Fragen:
    - Bietet die Universität Nachhilfe an?
    - Wo kann ich private Nachhilfe finden?
    - Welche Bücher werden empfohlen?
- BAföG
  - Fragen:
    - Was sind die Voraussetzungen für BAföG?
    - Wie lange kann ich BAföG bekommen?
    - Was passiert wenn ich die Regelstudienzeit nicht einhalte?

- Essen:
  - Fragen:
    * Wo kann man in der Universität essen?
    * Wann sind die Mensen geöffnet?

Es ist zwar sehr hilfreich, die Fragen einer größeren Menge von Studenten zu erfassen, diese Stichprobe entspricht aber doch noch nicht in allen Fällen der Fragen der Realität. Manche wichtigen Fragen können bei dieser Methode trotzdem aus Versehen ausgelassen werden. Das Ziel ist aber die sofortige Implementierung aller wichtigen Kernfragen.

Durch Absprache mit meinem Betreuer ist eine neue Idee entstanden. Diese besteht darin, die Mitarbeiter der Universität Düsseldorf zu befragen und dadurch einen neuen Fragenkatalog zu erstellen, welcher die in der Realität gestellten Grundfragen enthält. Dies ist besonders wichtig für den Zweck dieser Arbeit, denn genau solche Fragen sind schon des Öfteren gestellt worden und ähnliche beziehungsweise gleiche Fragen werden auch in Zukunft erwartet. Mit Hilfe von „google docs“ wurde eine Seite entworfen, die von jedem der Mitarbeiter editiert werden konnte. Im Folgendem befindet sich das fertige, durch Kooperation mit den wissenschaftlichen Mitarbeitern erstellte Dokument:

„Die Studierenden haben immer sehr viele, oftmals redundante Fragen. Bei jeder Frage, die nicht sofort selbst beantwortet werden kann, wird nach Antworten auf der Universitätsinternetseite gesucht. Falls dadurch kein Erfolg erzielt werden konnte, wenden sich die meisten an die Mitarbeiter der Universität Düsseldorf. Um die Anzahl dieser Fragen zu reduzieren, entwerfe ich (in meiner Bachelorarbeit) einen interaktiven Bot, der genau auf diese Standardfragen eingehen kann. Es wäre sehr hilfreich, wenn Sie diese Liste mit Fragen erweitern würden, die Ihnen oft von den Studienanfängern gestellt werden, damit wir ein Gespür dafür bekommen, auf welche Fragen der Bot eingehen können sollte. Lukas Miliunas (bitte die Liste ergänzen)

- Wann fangen welche Vorlesungen an?
- Welche Vorlesungen habe ich?
- Welche Anforderungen müssen in der Vorlesung XYZ erfüllt werden, um die Zulas-

sung zur Klausur zu erhalten?

- Wie melde ich mich zu einer Übungsgruppe der Vorlesung XYZ an?
- Wie sieht mein Stundenplan aus?
- Was für Programme sollte ich installieren?
- Kann ich auch mit Windows arbeiten?
- Welchen Laptop sollte ich kaufen?
- Was für Bücher brauche ich?
- Muss ich zu beiden Vorlesungen in der Woche oder ist das zweimal das gleiche?
- Wie kann ich mir Leistungen von anderen Unis/Berufsausbildung anerkennen lassen?
- Welche Nebenfächer kann ich wählen?
- Wie melde ich das Nebenfach an?
- Geht meine Klausurzulassung wieder verloren?
- Wie melde ich mich zur Klausur an/ab?
- Kann ich mich auch anmelden, wenn ich nicht zugelassen bin?
- Was mache ich, wenn ich zur Klausur krank bin?
- Wie oft kann ich eine Klausur wiederholen?“

Bei der Erstellung des Entwurfs für den Fragenkatalog haben ehemalige Studenten, Studierende und wissenschaftliche Mitarbeiter mitgeholfen. Jedoch ist dabei aufgefallen, dass noch eine besonders gute Möglichkeit ausgelassen wurde. Diese Variante besteht in einer Befragung von Erstsemestern. Daher wurde entschieden, diese während der ESAG Woche zufällig anzusprechen und folgende Fragen zu stellen: „Was würden Sie gerne noch erfahren, was Sie noch nicht wissen. Welche Probleme könnten eventuell beim Studienstart auftreten und

wären nicht direkt lösbar. Welche Fragen wären wichtig und benötigten sofort eine E–Mail–Nachfrage bei Mitarbeitern der Universität Düsseldorf?“ Durch dieses Verfahren wurden viele Fragen zusammengestellt. Es war schnell erkennbar, dass sich einige Fragen wiederholten und mehrere redundante Informationen erfragt wurden. Somit war es möglich, die Fragen zu gruppieren und zielführender zu definieren. Die richtigen Fragen auszusuchen, war nur der erste Teil des Gesamtprozesses. Die Suche nach Antworten ist etwas komplizierter. Manche Fragen bedürfen Antworten, die nur schwer im Internet auffindbar sind. Dennoch lässt sich ein Großteil der Fragen relativ leicht mit Hilfe des Internets beantworten. Daraus wird ersichtlich, dass nur wenige Fragen tatsächlich eine Nachfrage bei den Mitarbeitern erfordern.

Kapitel 4

Entwicklung des Chatbots

4.1 Dialogflow

Einen funktionsfähigen Chatbot zu entwerfen ist auch dann schwer, wenn alle Mittel dazu vorhanden sind. Um die ersten Schritte leichter zu machen, hilft insbesondere die Dokumentation von Dialogflow. Das Verständnis der Dokumentation ermöglicht es, Gedanken dazu zu entwickeln, wie die eigene Implementierung aussehen könnte. Um die ersten Erfahrungen leichter zu machen, stellt Dialogflow auf seiner Website einen implementierten Bot bereit, der lediglich „Hallo“ und nicht einmal „Wie geht es dir“ versteht. Als Bot–Sprache wird in diesem Projekt deutsch gewählt. Die Hauptkomponenten von Dialogflow sind Entities und Intents.

Entities [Dia18c] funktionieren wie Behälter, welche die benötigten Daten aus dem Gespräch mit dem aktuellen Benutzer extrahieren können. Durch mehrere Entwicklungsjahre sind bei Dialogflow sogar vordefinierte Entity Typen entstanden. Somit kann man in den Einstellungen sogar auswählen, dass zum Beispiel genau ein Datum, eine Zeit, ein Ort, oder sogar eine Farbe enthalten sein wird. Mittlerweile gibt es viele vordefinierte Entity–Typen [Dia18h], die in der Weiterentwicklung des Bots eventuell sehr hilfreich sein könnten. Da unser Bot eher spezifische Fragen beantworten soll, müssen benutzerdefinierte Entities eingebunden werden. Hier hat Dialogflow einen großen Vorteil, denn eine Erstellung des eigenen Entity–Typs ist vorgesehen und erfolgt sogar sehr einfach. Developer [Dia18b] Entities sind vom Benutzer auszufüllen. Der Entity–Block enthält ein Hauptwort (z.B. Haus). Dazu werden weitere Synonyme eingeführt (z.B. Häuser, Häuschen, Hütte). Durch dieses Verfahren können relativ

viele Wörter aus der natürlichen Sprache herausgefiltert werden. Dies stellt den wesentlichen Faktor dar, denn nur so wird ein umfassendes Verständnis davon erzeugt, was der Benutzer erfragen möchte.

Intents [Dia18f] definieren jeweils ein Gespräch und haben viele Attribute, welche die zu erwartenden Inhalte des Benutzergesprächs definieren können.

Der Intent hat vier Hauptkomponenten:

Intent Name Das ist der in der Dialogflow–Konsole erstellte Name für den aktuellen Intent, der auch in unserem Python–Programm sichtbar sein wird (der Name ist in dem Webhook enthalten).

Training phrases Das sind die Beispielfragen, die bei der Definition eines Intents hinzugefügt werden. Durch diese ist es möglich, die Hauptfälle abzudecken. Somit wird es für den Bot in der Zukunft leichter, ähnliche Fragen abzudecken.

Action and parameters Diese sind wichtige Komponenten. Action Name definiert den Namen von der Handlung, welche in dem Webhook unter Action angegeben wird. Dadurch kann das Python–Programm sofort erkennen, welche Action ausgelöst werden soll. Parameters sind entweder die vordefinierten Dialogflow–Entity–Typen oder Developer vordefinierte Entities. Diese können entweder optional oder required sein. Dieser Bereich ist wesentlich für die korrekte Funktion des Python–Backend.

Response Das ist die Antwort, die der Benutzer erhält.

Das Gespräch läuft (wie von Dialogflow vorgesehen) in drei Schritten ab.

1. Der Benutzer stellt Fragen, gibt also einen Input.

2. Dialogflow versucht die Eingabe zu parsen und zu verstehen. Sobald der Dialogflow–Agent den Input analysiert, wird er versuchen, diesen zu einem richtigen Intent zu matchen. Eventuell werden fehlende Informationen erneut bei dem Benutzer nachgefragt.

Falls die erwünschte Gesprächsrichtung erreicht wurde und ein Intent die benötigten Entities gematcht hat, kommt man zu einem in dieser Arbeit wesentlichen Zwischenschritt. Dieser besteht aus Fulfillment durch Webhook [Dia18d]. In diesem Fall werden anhand der Projektvoraussetzungen die extrahierten Informationen per Webhook direkt an den Server, der voraussichtlich auf Amazon Web Services laufen wird, geschickt. Dann wird das vorher beschriebene Python–Programm die Daten auswerten und dementsprechend die passende Antwort aussuchen. Die Antwort wird wiederum per Webhook zurück an Dialogflow geschickt.

3. Dialogflow liefert eine Antwort. Falls keine Intents ausgelöst wurden, wird eine Antwort aus „Default Fallback Intent“(eine vordefinierte „ich habe dich leider nicht verstanden“–Antwort) geliefert. Falls ein Intent doch ausgelöst wurde, existieren zwei weitere Fälle. Entweder sind die benötigten Entities in der Benutzerfrage sofort enthalten und Dialogflow kann den Webhook abschicken, oder es werden weitere Eingaben benötigt. In diesem Fall wird Dialogflow den Benutzer nach weiteren Eingaben fragen, bis die benötigten Daten eingegeben werden oder das Gespräch von dem Benutzer abgebrochen wird.

Diese beiden Komponenten (Entity und Intent) arbeiten effektiv miteinander. Die geschaffene Struktur ermöglicht dem Chatbot die Durchführung der gewünschten Gespräche. Das Framework ermöglicht, auf der Seite des Kunden, die Erzeugung einer natürlichen Konversation. Das Gespräch schafft außerdem ein Nebenprodukt: Die in den Nachrichten der Benutzer enthaltenen Informationen können extrahiert werden. Dadurch ist es möglich zu identifizieren, was der Benutzer erfahren möchte. Diese Informationen werden in einer JSON–Webhook–Datei gesammelt und können aufgrund der Fulfilment Funktion von Dialogflow an die gewünschte URL[1] zur Weiterverarbeitung der Daten gesendet werden. Solange die erwarteten Informationen nicht vollständig sind, können die fehlenden Informationen durch ein Intent–Attribut automatisiert nachgefragt werden.

Um zu verdeutlichen, wie ein einfaches Gespräch implementiert werden kann, wird im Folgenden ein Beispiel dargestellt. Zuerst wird ein Entity erstellt. Es heißt „Nebenfach“ und enthält die Werte „Nebenfach, Nebenfächer, Anwendungsfach, Anwendungsfächer“. Wie zuvor beschrieben, ist dieses Entity im Prinzip ein Objekt, welches bestimmte Wörter beziehungsweise Begriffe speichert.

[1] `https://www.omkt.de/url/`

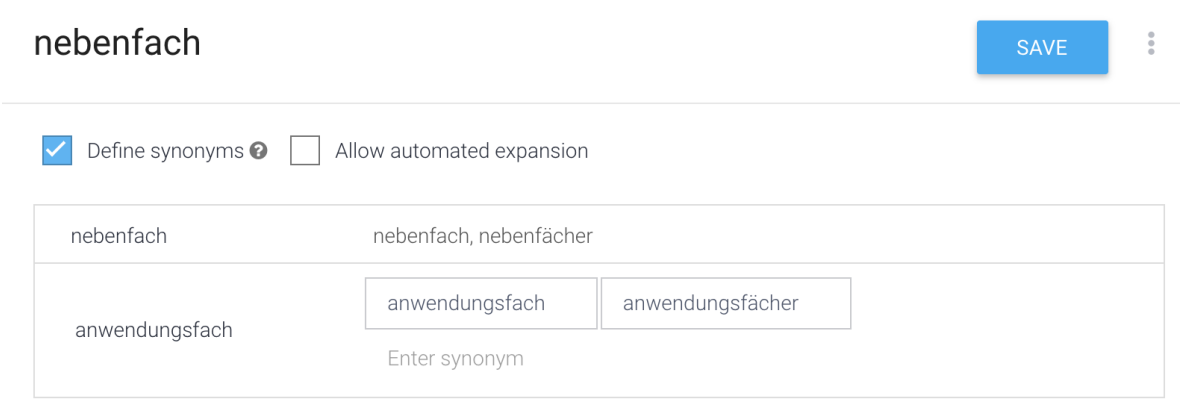


Abbildung 4.1: Entity-Erstellung auf der Dialogflow–Webseite

Anschließend wird ein Intent benötigt. Dieses trägt den Namen „nebenfach“ und benötigt unbedingt einen Parameter aus dem @nebenfach Entity. Ohne diesen Parameter wäre dieser Intent inaktiv.

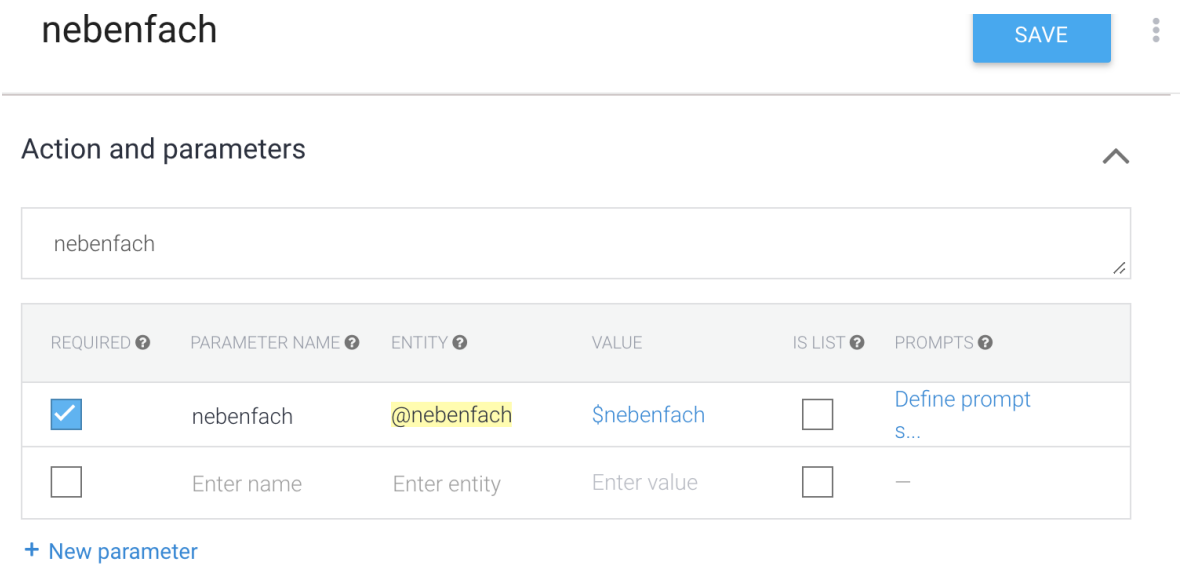


Abbildung 4.2: Auswahl der möglichen Required-Parameters auf der Dialogflow–Webseite

Danach wird eine Antwort benötigt, die unser Intent an den Benutzer übermitteln kann.

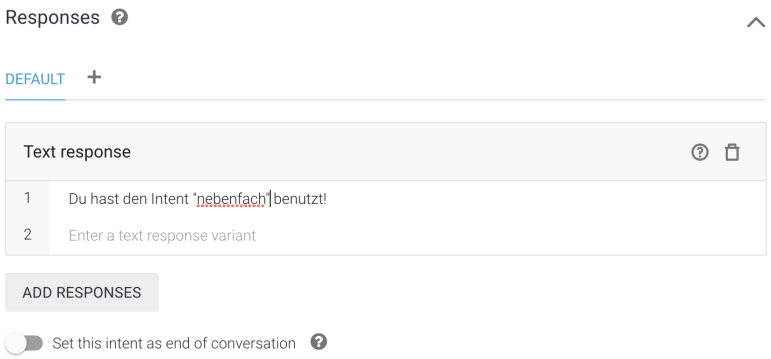


Abbildung 4.3: Definition einer Antwort auf der Dialogflow–Webseite

Im nächsten Schritt kann getestet werden, wie die Darstellung aus Benutzersicht ist, wenn folgende Frage gestellt wird:

„Ich möchte mehr über Nebenfächer (Informatik) wissen.“

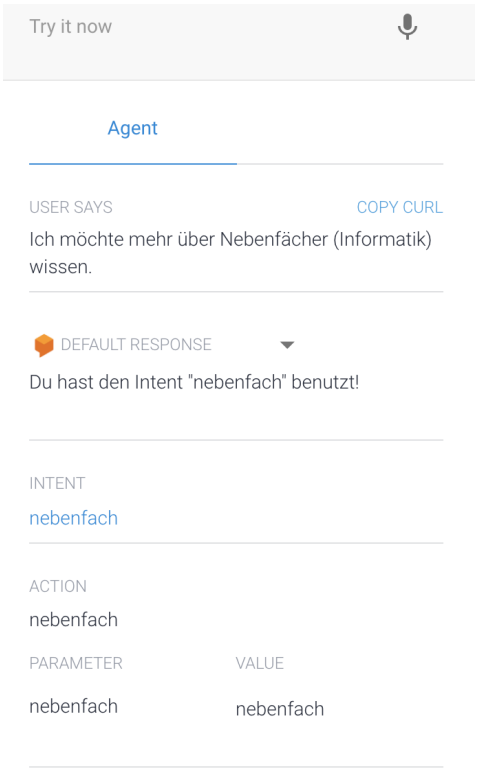


Abbildung 4.4: Beispiel einer Konversation mit dem Dialogflow–Bot

Somit ist klar, dass der Intent ausgelöst wurde und wir im Anschluss die erwartete Antwort erhalten haben. Theoretisch ist es also durch dieses Verfahren möglich, den Bot zu implementieren und zu benutzen. Die Funktionsweise ist aber immer noch sehr primitiv. Bei der Benutzung konnte der Intent nur eine einzige in Dialogflow vordefinierte Antwort liefern. Unser Stundentenbot bedarf eines komplexeren Lösungsweges. Dieser wird durch einen externen Server geschaffen, der die Funktionalität von Dialogflow deutlich verbessert.

Das letzte, was von der Dialogflow–Webseite berücksichtigt werden muss, ist Fulfillment [Dia18a]. Dieses Objekt erstellt einen HTML POST Request und füllt es mit Informationen, welche aus Benutzergesprächen extrahiert werden. Die einzigen Schritte, die hier berücksichtigt werden müssen, sind folgende: Webhook muss auf der Dialogflow–Webseite eingeschaltet werden und unter der URL muss die Adresse des Servers eingetragen werden, der die Webhooks erwartet.

4.2 Public Cloud

Um die Entwicklung fortzusetzen, ist die Benutzung eines passenden Backends notwendig. Wie schon vorhin angekündigt, wird die Webhook–Zusammenstellung in einem Python–Programm ablaufen. Aber zuerst muss eine Instanz erstellt werden, die als Server funktioniert. Die bei Amazon Web Services angebotene EC2 [Ama18a] ermöglicht genau das. Anbei wird eine EC2–Intstanz [Ama18g] erstellt. Als Betriebssystem wird Ubuntu 16.04.4 LTS gewählt. Gründe für diese Wahl sind insbesondere die kostenlose Nutzung, sowie Installationsmöglichkeiten auf einer AWS–Intstanz. Ein weiterer Grund ist, dass dieses Betriebssystem bereits in mehreren Universitätsmodulen benutzt wurde und daher bekannt ist. Desweiteren muss ein Intstanztyp [Ama18c] gewählt werden. Hierbei ist „t2.micro“ zu präferieren, da die Ressourcen ausreichend sind und diese in den am Anfang bereits erwähnten „free–trier“ Rahmen hereinpassen. Als nächstes wird ein SSH–Schlüssel erzeugt. Dieser wird benötigt, sobald man eine SSH–Verbindung [SSH18] mit der Instanz aufbaut.

Die Instanz ist nun erzeugt. Um jedoch die Instanz von Dialogflow erreichen zu können, müssen weitere Konfigurationen durchgeführt werden. Zuerst ist ein Netzwerk zu erzeugen. Aus der Sicht Amazons muss ein VPC [Ama18e] erstellt werden. Die Komponente „Name tag“ enthält einen Namen von unserem VPC, welcher frei wählbar ist. Die andere Komponente ist IPv4 CIDR block. Hier muss eine IP–Adress–Range eingegeben werden. Unsere Instanz hat

eine von AWS[2] zugewiesene private IP–Adresse 172.31.32.35. Daher wird 172.31.0.0/16 als IP–Range gewählt.

Das Netzwerk ist erzeugt. Um es jedoch erreichbar zu machen, wird eine Security Group [Ama18d] benötigt. Security Groups definieren, welche IP–Adressen unter welchen Ports freigegeben werden. Der ausgehende Verkehr ist laut Default–Einstellungen immer freigegeben. Der eingehende Verkehr ist blockiert. Daher muss eine Security Group erzeugt werden, die sowohl den Verkehr durch SSH (Inbound Regel: Protokoll: TCP, Source: 0.0.0.0/32, Ports:22), als auch diesen durch HTTP (Inbound Regel: Protokol: TCP, Source: 0.0.0.0/32, Ports:80) erlaubt. Dies sind die einzigen Regeln, die wir in der Sicherheitsgruppe bei diesem Projekt benötigen werden.

Jetzt fehlt nur noch ein letzter Schritt: Die Instanz muss aus dem Internet erreichbar sein. Zur Zeit hat sie nur eine private IP–Addresse. Um die Instanz von Dialogflow erreichen zu können, wird eine Public–IP [Ama18b] benötigt. Während der Instanzerzeugung kann die Option „Auto assign public IP“ gewählt werden. In der AWS–Konsole, unter „EC2—->Running Instances“, können alle Informationen über die Instanz jederzeit nachgeschaut werden:

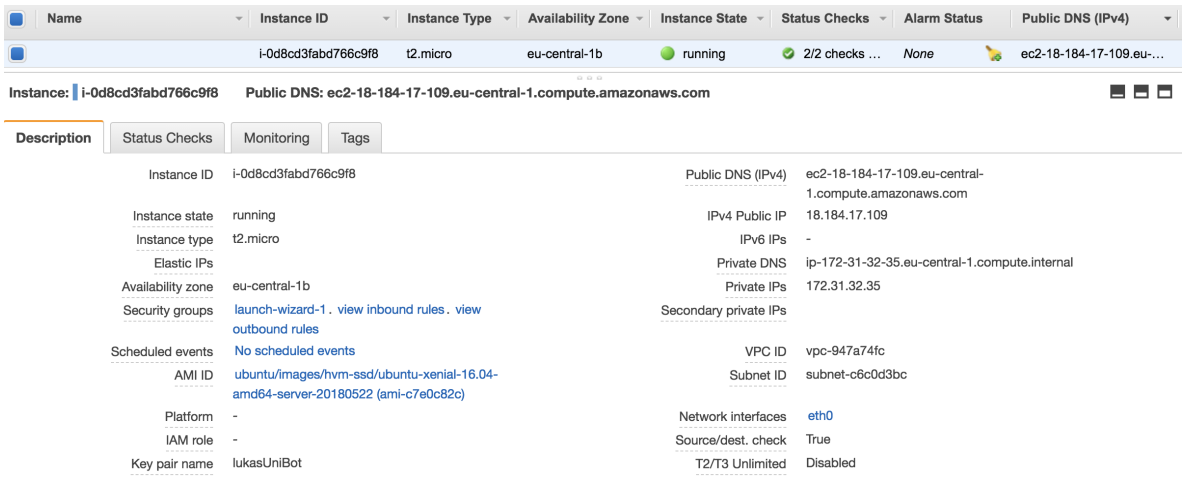


Abbildung 4.5: Ansicht einer EC2-Instanz auf der Amazon–Web–Services–Webseite

[2]Amazon Web Services

Nun sind die Umgebungen fertig. Jetzt soll unser Python–Programm konfiguriert werden. Alle Skripte, die in dieser Bachelorarbeit benutzt werden, sind immer mit Version–Control–Gitlab–Repository gesichert.

4.3 Python–Response

Um die Funktionalität des Bots zu prüfen, wird zunächst für den vorher getesteten Intent „Nebenfach“ die Antwort mit Python erstellt. Die Aufgabe des Python–Programms ist es, die richtige Antwort zurück an Dialogflow zu liefern. Daher entsteht eine neue Problematik, welche darin besteht, den besten Platz für die Lagerung der Bot–Antworten zu finden. Eine einfache Lösung wäre die direkte Abspeicherung der Antworten in einem Python–Skript. Dies wäre jedoch auch extrem ineffizient. Es muss berücksichtigt werden, dass die Antworten viele universitätbedingte Informationen sowie viele Internetadressen enthalten. Das bedeutet, diese müssen ständig gepflegt werden. Falls jedes Mal bei der Bearbeitung von Antworteninformationen ein Login in der Instanz und das Bearbeiten des Python–Codes benötigt würden, wäre diese Lösung übertrieben umständlich. Durch Absprache mit meinem Betreuer und kurzes Überlegen sind wir uns einig geworden, dass zur Zeit ein Gitlab–Snippet [Git18] die richtige Lösung ist. Unser Gitlab–Snippet wird zunächst nur eine einzige Zeile beinhalten: „nebenfach: Die Informationen über das Nebenfach und die Anmeldung hierfür kannst du auf der folgenden Seite finden —-> https://www.cs.hhu.de/studium–lehre–informatik/studierende/nebenfaecher.html“

Die Antworten können somit alle in dem Gitlab–Snippet hinzugefügt und an einer Stelle schnell und unkompliziert modifiziert werden. Um die Antworten mit Python–Skript zu erreichen wird eine Gitlab–Library [Gau18] verwendet. Wir erzeugen eine Python–Datei „answers.py“. Anbei ist ein Beispiel, wie man sich mit dem Gitlab verbindet:

```
def create_gitlab_connection():
    try:
        gl = gitlab.Gitlab('https://gitlab.cs.uni-duesseldorf.de/', "GitlabAuthPassword")
        gl.auth()
    except Exception as e:
        print("Could not connect to gitlab. Error message : " + str(e))
        return
    return gl
```

Abbildung 4.6: Gitlab–Authentifizierung im Python–Programm

Sobald eine Möglichkeit besteht die Verbindung mit Gitlab herzustellen, wird der Snippet im Python–Skript geladen und in einem Dictionary–Objekt gespeichert:

```
def load_snippet():
    gl = create_gitlab_connection()
    snippet = gl.snippets.get(21)
    content = snippet.content().decode()
    my_dict = yaml.load(content)
    return my_dict
```

Abbildung 4.7: Herunterladen von Gitlab–Snippet im Python–Programm

Um die Funktionalitätsvorschriften von einem Webhook in Dialogflow einzuhalten, müssen wir unsere Antwort zuerst mit einem korrekten Kennwort kennzeichen. Laut der Dokumentation [Dia18e] ist dies „fulfillmentText“. Wir erstellen noch eine Python–Datei – „intent_functions.py“. Die dazugehörge Python–Funktion würde wie folgt aussehen:

```
def make_response(subject: str):
    return {"fulfillmentText": subject}
```

Abbildung 4.8: Erstellung einer richtigen Python–Antwortstruktur

Zunächst wird eine Funktion gebraucht, welche die vollständige Antwort liefert. Diese Antwort kann sofort an den Dialogflow–Agent zurückgeschickt werden:

```
def nebenfach():
    return make_response(my_dict.get("nebenfach"))
```

Abbildung 4.9: Definition der Python–„nebenfach“–Funktion

Es ist erkennbar, dass eine spezifische Funktion vorliegt, welche einen einzigartigen Namen trägt. Diese Funktion wurde also nur für einen Intent erzeugt. Das heißt, diese bestimmte Funktion wird genau dann aufgerufen, wenn der genaue Intent angefragt wird. Da wir zukünftig sehr viele ähnliche Funktionen mit verschiedenen Namen haben werden, ist es sehr wichtig, sofort ein Dictionary für diese Funktionen zu erzeugen. Somit wird in der Weiterentwicklung sehr viel Laufzeit gespart und die Antwort an Dialogflow kann stabiler gestaltet werden. Das Dictionary heißt „actions“ und wird in einer „actions.py“–Datei gespeichert:

```
actions = {
    "nebenfach": nebenfach()
}
```

Abbildung 4.10: Definition des Dictionarys in Python

Jedes Mal, wenn eine neue Funktion durch einen neuen Intent entstehen wird, wird sie zu diesem Dictionary hinzugefügt. Das Objekt „actions“ wird also als Hauptstelle fungieren, um den passenden Webhook herzustellen.

Wenn wir annehmen, dass wir einen Webhook von Dialogflow erhalten hätten, sollte dieser zuerst bearbeitet werden, um den gefragten Parameter zu identifizieren.

Nun erfolgt noch die Erstellung der Python–Datei „dialog.py“. Die dabei enthaltenen Funktionen werden als Programmanstoßpunkt verwendet. Dafür ist es notwendig, eine weitere Funktion zu definieren, die den erhaltenen Webhook bearbeitet:

```
def processRequest(req):
    requestblock = req.get("queryResult")
    action = requestblock.get("action")
    parameters = requestblock.get("parameters")
    return actions.get(action)
```

Abbildung 4.11: Bearbeitung des erhaltenen Webhooks in Python

Die in dieser Funktion durchgeführten Schritte sind trivial, wenn man den Inhalt [Dia18e] des Beispiels request–Webhooks von Dialogflow betrachtet. Der Webhook könnte nicht nur Action, sondern auch einige weitere Parameter enthalten. Dies sprengt aber den Rahmen dieser Bachelorarbeit. Die Benutzung anderer Parameter als Action wäre erst bei einer möglichen Weiterentwicklung zu betrachten, falls man an einer komplexeren Funktionalität des Bots interessiert wäre. Wie man diese genau benutzen könnte, wird im Kapitel „Fazit und Ausblick“ erläutert. In dieser Arbeit wird nur der Parameter Action verwendet.

Die letzte Zeile in dieser Funktion liefert einen Wert aus dem zuvor definierten Dictionary. In diesem Dictionary wird sofort eine entsprechende Funktion aufgerufen und die Antwort von dem Gitlab–Snippet geliefert.

Ein wesentlicher Schritt ist es nun jedoch, den Dialogflow–Webhook tatsächlich zu erhalten. Um einen HTTP Webhook zu bekommen, muss unser Python–Programm als ein Server funktionieren und die ganze Zeit auf Port 80 (HTTP default Port) zuhören. Um das zu erreichen wird Flask [Arm18a] benutzt. Flask funktioniert [Arm18b] wie ein Webserver. Die genaue Definition, die in diesem Projekt benutzt wird, läuft auf „0.0.0.0“ und hört auf Port 80 zu:

```
app = Flask(__name__)

if __name__ == '__main__':
    port = int(os.getenv('PORT', 80))
    print("Starting app on port %d" % port)
    app.run(host='0.0.0.0', port=port, debug=True)
```

Abbildung 4.12: Definition eines Python–Flask–Servers

Als letztes wird eine Funktion benötigt, die tatsächlich die Webhooks aus Dialogflow abfängt. Aus einem öffentlichen Beispiel [art18] kann man die Funktionalität erkennen und eine Implementierung erzeugen:

```
@app.route('/webhook', methods=['POST'])
def webhook():
    req = request.get_json(silent=True, force=True)
    res = processRequest(req)
    res = json.dumps(res, indent=4)
    r = make_response(res)
    r.headers['Content-Type'] = 'application/json'
    return r
```

Abbildung 4.13: Abfangen der Webhooks aus Dialogflow in Python

Die vorher von Amazon Web Services beantragte Public–IP[3] kann jetzt benutzt werden. Jedes Mal, wenn ein Webhook an die URL [4] geschickt wird, wird es auch von der Funktion „Webhook“ abgefangen und eine Antwort wird zurückgeliefert.

Damit die Implementierung abgeschlossen ist, muss zunächst die Webhook–Funktion in der Dialogflow–Konsole aktiviert und die richtige URL eingetragen werden.

[3] 18.184.17.109
[4] http://18.184.17.109/webhook

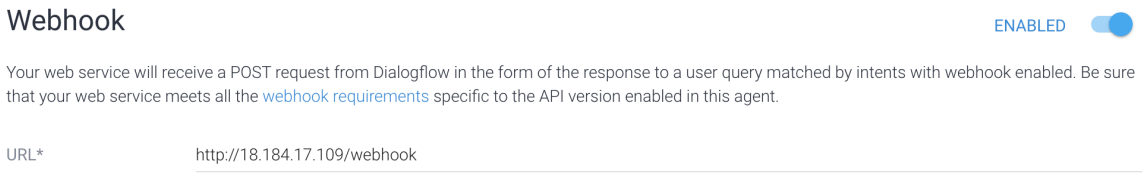


Abbildung 4.14: Aktivierung des Webhooks in der Dialogflow–Konsole

Auch in dem „Nebenfach“–Intent muss die Fulfillment–Funktion aktiviert werden:

Abbildung 4.15: Aktivierung des Dialogflow–Fulfillments in der Dialogflow–Konsole

Nun wird das dialogtest.py–Programm gestartet und die vorherige Frage „Ich möchte mehr über Nebenfächer (Informatik) wissen.“ erneut gestellt:

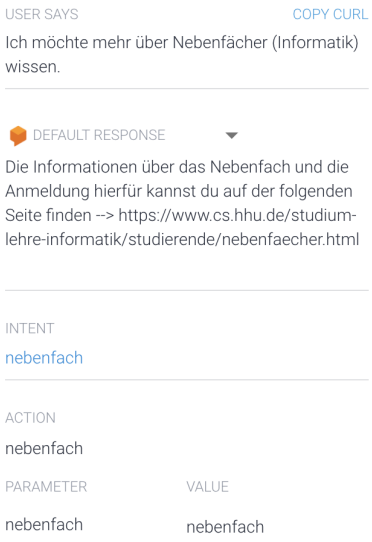


Abbildung 4.16: Ansicht der Konversation mit dem Dialogflow–Bot nach der Fulfillment–Aktivierung

Die erhaltene Antwort deutet darauf hin, dass unsere Anwendung funktioniert, da die Antwort richtig aus dem Gitlab–Snippet geladen wurde.

Alle weiteren Intents werden analog zu diesem Beispiel sowohl in der Dialogflow–Konsole als auch in Gitlab–Snippet und in Python–Dateien eingetragen. Alle erzeugten Intents und die dazugehörigen Antworten sind in Gitlab–Snippet und im Python–Code ersichtlich.

Es ist nicht der Sinn eines Servers, ständig ein Command Line Programm offen zu halten. Dieses sollte selbstständig laufen. Daher wird ein Service [Can18] definiert, welcher unsere dialog.py–Datei ausführt.

Das Python–Programm soll auf dem Server laufen. Es wird angestrebt, dass jegliche Änderungen leicht durchführbar sind. An dieser Stelle wird Ansible [Red18] eingeführt. Ansible ist ein Werkzeug, welches es ermöglicht, die Automatisierung in der Cloud einfacher und eleganter durchzuführen. Es erschafft die Möglichkeit, die gewünschten Aufgaben remote per SSH auszuführen. Ansible–Task wird automatisiert mit Gitlab–Continuos–Integration ausgeführt. Die genauen Definitionen und Beschreibungen von Ansible und Continuous–Integration sind in einer README.MD–Datei bei den Code–Dateien beschrieben. Die Konfiguration ist somit abgeschlossen und die in dieser Arbeit beschriebene Anwendung ist bereit.

Der nächste Schritt ist, analog zu dem beschriebenen „Nebenfach“–Intent–Beispiel, alle Frage–Antwort–Varianten aus dem bereits vorhandenen Fragenkatalog und aus der Gitlab–Anworten–Sammlung zu der Anwednung hinzuzufügen und den Bot vollständiger zu gestalten.

Dieser Teil ist tatsächlich sehr anspruchsvoll. Es ist nicht ganz einfach zu entscheiden, welche Fragen am besten gruppiert werden sollten und welche eigene Intents ansprechen. Auch die Antworten bedürfen viel Zeit. Das sind zum größten teils die Fragen, welche die Erstsemester bereits an die Mitarbeiter der Heinrich–Heine–Universität geschickt haben. Das deutet darauf hin, dass diese Antworten nicht alle von alleine im Internet oder ähnliches gefunden werden können. Alle in dieser Bachelorarbeit entstandenen Antworten sind durch Absprache mit meinem Betreuer entstanden, sowie über Internetseiten, die mit der Universität in Verbindung stehen. Die Antworten sind alle in dem Gitlab–Snippet in „miliunas–code“ Gitlab Repository zu finden.

Alle nachfolgenden Fragen, Antworten, Intents, Entities und Konfigurationen bedürfen keiner ausführlicheren Beschreibung, weil diese im Prinzip analog zu dem „Nebenfach“ Intent erzeugt wurden und lediglich der Inhalt verändert wurde. Nach diesem Schritt scheint es, als ob der Bot fertig wäre. In der Realität stößt man jedoch sofort auf ein Problem. Viele Fragen werden nicht abgefangen oder sogar falsch verstanden.

4.4 Training

Eine Hilfe an dieser Stelle ist das Training [Dia18i] in der Dialogflow–Konsole. Alle internen und externen Fragen, die diesem Bot schon gestellt wurden, sind unter „Training“ in der Dialogflow–Konsole auffindbar. Dabei kann man bei jeder Frage sehen, ob ein Intent gematcht wurde und anschließend entscheiden, ob der Match richtig ist. Wenn dieser falsch ist, dann kann der richtige Intent zugewiesen werden. Beim erneuter Anfrage wird der Bot auf die gleiche Frage die richtige Antwort liefern, da er dies zuvor im Training gelernt hat.

In dem ganzen Entwicklungsprozess wurde das Training–Feature mehrere hundert Male benutzt, um immer wieder neue „Training Phrases“ und aktuelle Konversationen zu prüfen und es zu evaluieren. Somit ist die grundlegende Entwicklung beendet.

4.5 Webpage-Integration

Um die Entwicklung abzuschließen wäre es wünschenswert, den Bot nicht nur in der Dialogflow–Demo–Version, sondern durch Implementierung auf einer Webseite nutzen zu können. Eine mögliche Lösung wäre der Bau einer eigenen Integration. Dieser Ansatz wäre jedoch zu anspruchsvoll, um diesen im Rahmen der Bachelorarbeit noch zu begehen. Daher erfolgte die Suche nach einer möglichen Implementierung. Dabei wurde ein Integrationsvorschlag gefunden [**?**], welcher durch Kommunicate [Kom18] erfolgt. Zuerst wird der Dialogflow–Identitäts–Key bei Kommunicate hinterlegt, damit die Verständigung zwischen Kommunicate und dem Dialogflow–Bot möglich ist. Die Dialogflow–Integration in Kommunicate erfolgt dann sogar nur durch einen Klick:

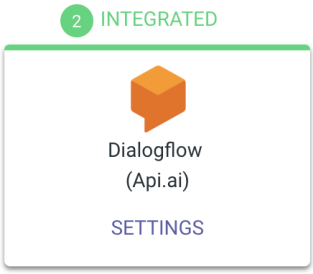


Abbildung 4.17: Dialogflow–Integration auf der Kommunicate.io–Webseite

Um diese Chatfunktionalität auf einer Webseite bereitzustellen, bietet Kommunicate ein Java–Skript an. Dieses soll vor dem `<head>` in die HTML Datei der Webseite hinzugefügt werden:

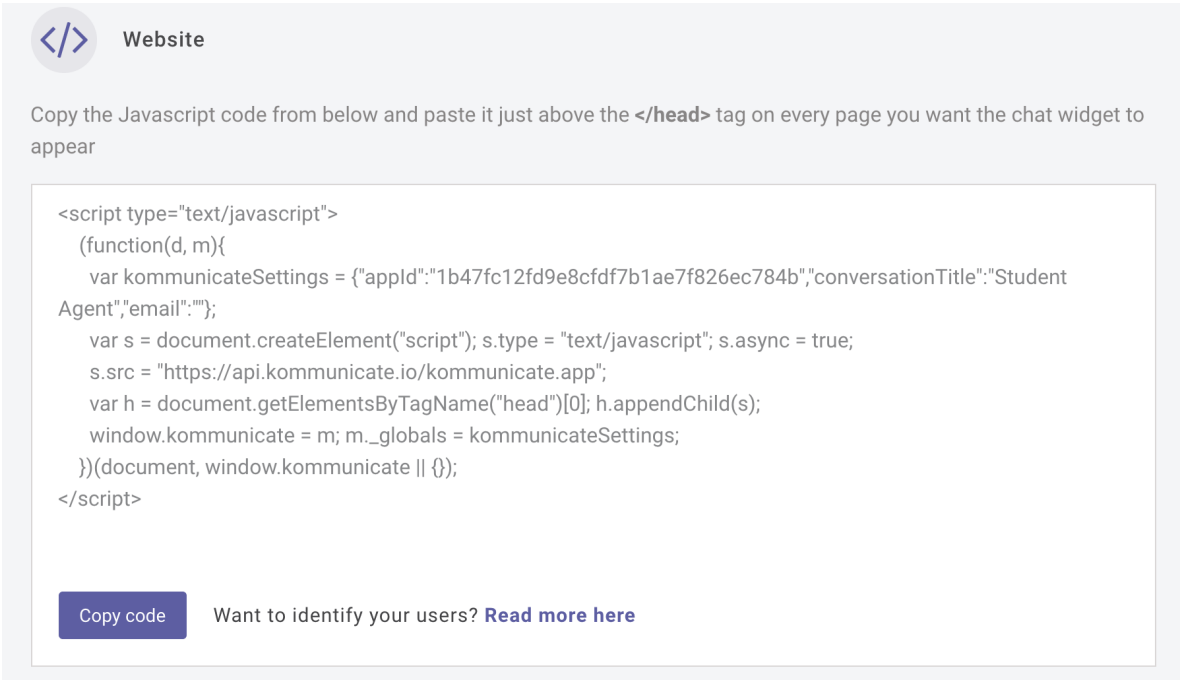


Abbildung 4.18: HTML–Code zur Aktivierung der Kommunicate.io-Integration auf der eigenen Webseite

Diese Integration soll nun im nächsten Schritt ausprobiert werden. Dafür benötigen wir eine Webseite. Als Werkzeug wird „nginx“ [NGI18] gewählt. Dieses Modul bietet die Möglichkeit, einen statischen Webserver zu starten, der eine einzelne index.html–Datei unter dem gewünschten Port freigibt. Dabei wird Port 81 sowie unsere Public–IP–Adresse benutzt. Der Grund für die Benutzung des Port 81 ist, dass Port 80 bereits von Dialogflow benutzt wird und wir aus Kostengründen nur eine Instanz weiterhin benutzen möchten. Die Datei index.html wird mit dem Inhalt aus einer kostenlosen, angepassten HTML–Template [Fre18] gefüllt. Genau bevor dem <head> wird der eben beschriebene Code von Kommunicate hinzugefügt. Durch diese Schritte wurde eine Seite entworfen, die unter die URL [5] erreichbar ist und wie folgt aussieht:

Abbildung 4.19: Ansicht der erstellten Webseite mit der Bot–Integration

[5] „18.184.17.109:81“

Sobald auf die Sprechblase rechts gedrückt wird, erscheint eine Dialogbox und der Bot ist bereit für die Kommunikation:

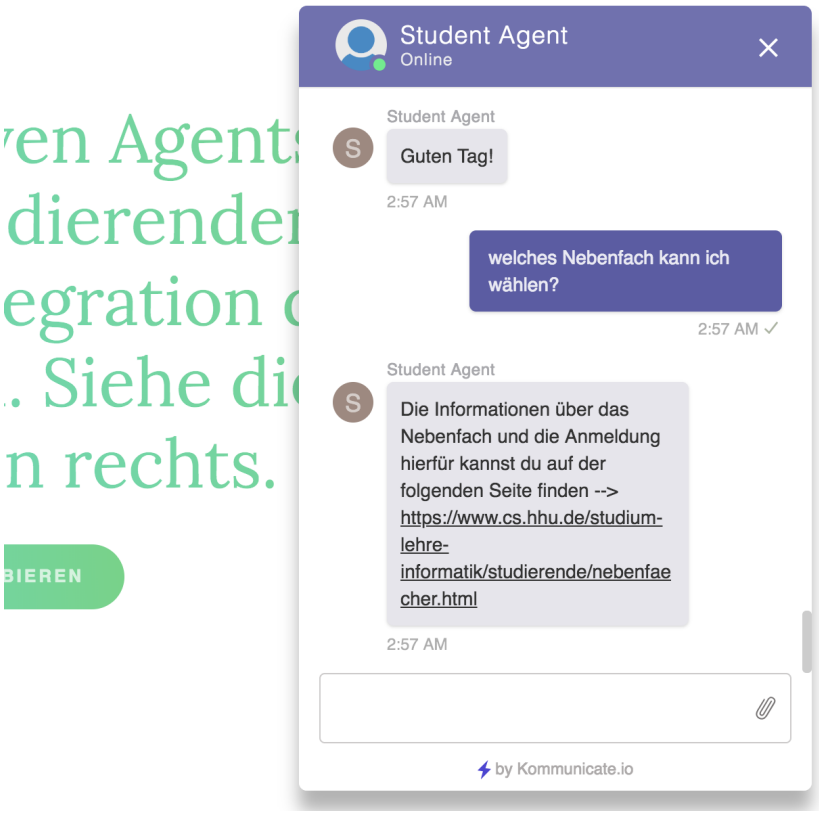


Abbildung 4.20: Beispiel einer Konversation mit dem Bot auf der Webseite

Alternativ kann immer noch die Web-Demo-Version des Chatbots unter „https://bot.dialogflow.com/26ac65dd-b9b1-41d6-812e-2d41729ef5de“ benutzt werden.

Kapitel 5

Testing

Es ist sehr wichtig festzustellen, ob der Dialogflow–Bot so wie erwartet funktioniert. Training–Phrases sind dabei sehr hilfreich. Das Durchschauen der gematchten Entities ermöglicht die Verbesserung des Bots. Dies könnte aber immer noch nicht als Testing bezeichnet werden.

Dialogflow–Bots sind in der ganzen IT–Umgebung eine relativ neue Anwendung. Daher ist es schwer, ein funktionierendes Testing–Framework zu finden. Eines selber zu erstellen wäre eine Lösung, jedoch würde dies zu viel Zeit im Anspruch nehmen. Ständiges Durchsuchen und Ausprobieren hat ergeben, dass es eine bessere Lösung gibt. Diese heißt Botium [Den18].

Florian Treml (Gründer von Botium) hat selber einen Artikel [Flo18] veröffentlicht, welcher beschreibt, wie man das Dialogflow–Testing einstellen kann. Nach der erfolgreichen Installation muss man sich mit einem Google Account identifizieren und die Authentifizierungsdaten in einer „botium.json“–Datei in dem aktuellen Verzeichnis abspeichern. Die genauere Authentifizierungsstruktur ist auf der oben benannten Seite dokumentiert.

Beim weiteren Vorgehen werden mit dem Befehl „botium–cli import dialogflow–intents“ die bereits vorhandenen Dialogflow–Intents heruntergeladen. Die schon vorher in Dialogflow benutzten Eingaben, die diese Intents ausgelöst haben, sind bereits im Botium–Import enthalten. Anbei werden automatisch folgende Dateien erzeugt: „IntentName.convo.txt“ und „IntentName_input.utterances.txt“. Die erste Datei enthält die Struktur einer Konversation. Wenn dabei das gleiche Beispiel mit „Nebenfach“–Intent betrachtet wird, sieht die Datei wie folgt aus:

```
nebenfach

#me
nebenfach_input

#bot
nebenfach
```

Abbildung 5.1: Die Gesprächsstruktur der Botium–Plattform

Zuerst wird der Intent–Name angegeben. Die Zeile nach „#me“ deutet auf die Eingabe hin. In diesem Fall wird nach möglichen Eingaben in „nebenfach_input.utterances.txt“ in dem gleichen Verzeichnis gesucht. Die Datei „nebenfach_input.utterances.txt“ enthält bereits folgende Eingaben:

```
nebenfach_input
Wie kann ich mich für mein Nebenfach anmelden?
Kann ich BWL als Nebenfach wählen?
Bei wem kann ich mich für mein Nebenfach anmelden?
Bis wann kann ich mich für das Nebenfach anmelden?
Ich möchte mehr über Nebenfächer (Informatik) wissen.
```

Abbildung 5.2: Ansicht einer Botium–Input–Datei

Mit der Zeit werden neue Fragen von möglichen Bot–Benutzern gestellt. Beim weiteren „botium–cli import dialogflow–intents“ werden alle bis dahin zu diesem Intent gestellten Fragen bereits in dieser Input–Datei vorhanden sein.

Die Zeile „#bot“ deutet auf eine Bot–Antwort hin. In diesem Beispiel haben wir nur eine Frage von „#me“ und eine Antwort von dem Bot. Man kann dieses Gespräch mit weiteren „#me“ und Bot–Zeilen erweitern, falls der Bot in der Zukunft weiterentwickelt würde. Die möglichen Bot–Antworten werden in der „nebenfach_output_0.utterances.txt“–Datei erwartet. Die Datei existiert aber noch nicht und muss erzeugt werden. Dies wird durch ein Python–Skript ermöglicht:

```
def create_answer_files():
    with open("/dialogflow/botium/nebenfach_output_0.utterances.txt", 'w+') as f:
        f.write("ulb" + '\n')
        f.write(my_dict.get("nebenfach") + '\n')
        f.write('\n')
```

Abbildung 5.3: Python–Code zur Erstellung von Botium–Antworten

Mit dem Befehl „botium–cli run“ werden alle möglichen „convo“–Dateien im aktuellen Verzeichnis getestet. Im „Nebenfach“ Fall wird jede Frage aus „nebenfach_input.utterances.txt“ an den Dialogflow–Bot verschickt und die Antwort mit den Daten in „IntentName_output_0.utterances.txt“ verglichen. Hier ist das Testergebnis:

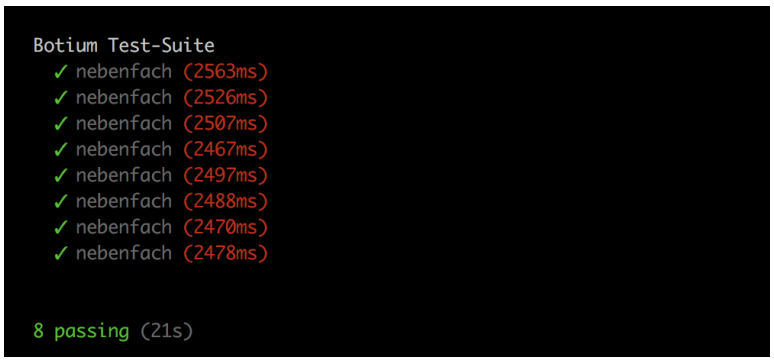


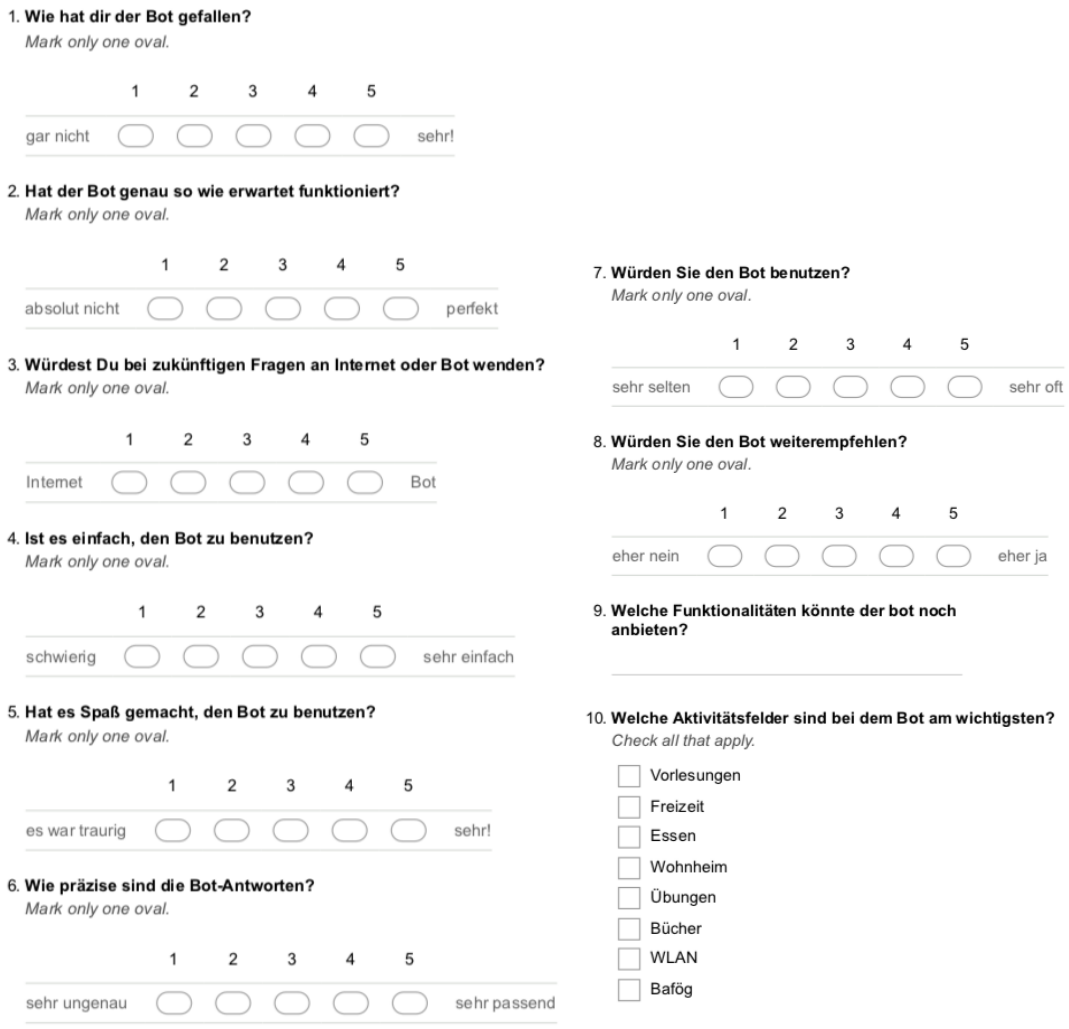
Abbildung 5.4: Erfolgreicher Durchlauf von Botium-Tests

Alle acht zur Zeit vorhandenen Eingabefragen in „nebenfach_input.utterances.txt“ haben die richtige Antwort von Dialogflow erhalten. Analog können alle anderen Intents getestet werden. Das Testverfahren könnte auch in Continuous Integration Workflow integriert werden. Das würde aber bei jedem Durchlauf deutlich zu viel Zeit im Anspruch nehmen. Aus dem oberen Bild kann entnommen werden, dass alleine dieser Intent um die 20 Sekunden Zeit im Anspruch genommen hat. Die Tests werden sequentiell ausgeführt. Botium–cli bietet leider keinen parallelen Aufruf. Daher wird das Testing, falls gewünscht, remote durch Command Line Interface ausgeführt. Somit kann jederzeit ausprobiert werden, ob der Dialogflow–Bot wie erwartet funktioniert. Ein solches Testingverfahren erleichtert die Weiterentwicklung.

Kapitel 6

Evaluation

Ob der Dialogflow–Bot richtig und benutzerfreundlich funktioniert, kann nicht durch eine einzige Person entschieden werden. Zuerst wurde mit Hilfe zweier Mitstudenten ein Pretest durchgeführt, indem dem Bot mögliche Fragen gestellt wurden. Dabei sind einige gewöhnliche Fehler aufgetreten. Diese konnten aber mit Bot–Training beseitig werden. Dabei ist auch klar geworden, dass jedes Mal bei der Erstellung eines neuen Intents so viele Testfragen wie möglich gestellt werden sollten. Dadurch werden immer mehr Fragen erkannt und das Training des Bots wird leichter. Um eine generelle Rückmeldung über die Funktionalität des Bots zu erhalten, sollte eine Umfrage diesbezüglich erstellt werden. Eine Evaluation wäre also sehr sinnvoll, insbesondere da es sich um eine Anwendung handelt, die in der Zukunft möglicherweise von mehreren Leuten benutzt wird. Aus diesem Grund wurde folgende Umfrage erstellt:



(a) Seite 1 (b) Seite 2

Abbildung 6.1: Ansicht des Evaluationsbogens

Sofort beim ersten Feedback wurde festgestellt, dass der Bot noch ein nicht beseitigtes Problem aufweist. Wenn die Benutzerfrage keinen Intent gematcht hat, sollte richtigerweise eine Antwort aus „Default Fallback Intent“ zurückgegeben werden. Zum Beispiel „Ich verstehe dich leider nicht oder diese Funktion ist noch nicht implementiert“. Bei manchen Gesprächen, bei denen kein Intent gematched wurde, vermutet Dialogflow anhand mancher Teilwörter, dass ein Intent vielleicht doch zu der aktuellen Frage das richtige wäre. Falls dieser vermuteter Intent „required parameter“ besitzt, wird Dialogflow eine Default Nachfrage „What is the <entity name> ?“ zurückliefern. (Die Default–Nachfrage erfolgt auf english, obwohl der Bot auf deutsch eingestellt ist). Ab diesem Punkt erwartet Dialogflow–Agent entweder einen

Parameter aus diesem bestimmten Entity oder das Wort „abbruch“, welches dieses Gespräch abbrechen würde.

Ein Beispiel des beschriebenen Problems:

- Benutzer: Darf ich Rechnernetze hören?
- Bot: What is the wlan?
- Benutzer: Ich möchte die Vorlesung Rechnernetze hören, wer hält sie?
- Bot: What is the wlan?
- Benutzer: netze netze netze?
- Bot: Wie du das WLAN einrichten kannst findest du unter https://www.zim.hhu.de/services–des–zim/netz/netzzugang/wifi–wireless–lan.html

Der Bot hat in diesem Fall zuerst die falschen Fragen gestellt, anschließend aber auch eine falsche Antwort gegeben. Das genaue Problem, welches hier passiert ist: Die Frage „Darf ich Rechnernetze hören?“ hat keinen Intent gematcht, da das Thema Rechnernetze von dem Bot noch nicht gedeckt wurde. Aber anhand des Teilwortes „netze“ (Rechnernetze) hat der Bot vermutet, dass der Benutzer eventuell etwas über das Thema WLAN erfahren möchte, denn der Parameter „netze“ ist im WLAN Entity enthalten. Daher fragt der Bot „What ist the wlan?“ und erwartet entweder eine Bestätigungsantwort mit dem richtigen Parameter oder das Keyword „abbruch“.

Dieses Verfahren ist aber für den Benutzer nicht verständlich. Damit ist der Bot aus Benutzerseite nicht nutzbar, da er ständig in Deadlocks läuft. Es war nicht sofort klar, wie ein solches Problem beseitigt werden soll. Nach einer längeren Recherche wurde jedoch festgestellt, dass bei jedem Required Entity eine Frage dazu definiert werden kann, die den Benutzer über den aktuellen Status informiert. Das kann als „Prompt“ eingetragen werden:

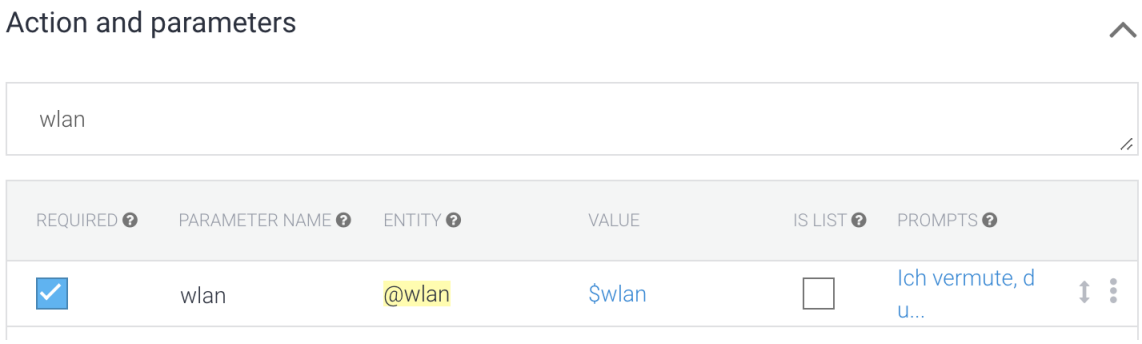


Abbildung 6.2: Erstellung der Dialogflow–Prompt–Kontrollfragen

Nach diesem Update sieht das eben beschriebene Gespräch folgendermaßen aus:

- Benutzer: Darf ich Rechnernetze hören?
- Bot: Ich vermute, du möchtest mehr über WLAN erfahren. Falls ja – schreib „wlan“, falls nein – „abbruch“.
- Benutzer: abbruch
- Bot: Abgebrochen.

In diesem Fall hat der Benutzer aufgrund des fehlenden passenden Intents zwar keine Informationen erhalten, aber immerhin ist das Deadlock–Problem gelöst und der Bot kann weiterbenutzt werden. Durch dieses Gespräch wurde auf einen fehlenden Intent aufmerksam gemacht. Dieses ist mittlerweile implementiert worden.

Ergebnisse der aktuellen Evaluation: Mehr als 50 Studierende und wissenschaftliche Mitarbeiter wurden gebeten, den Bot zu evaluieren. Davon haben sich 16 Leute den Bot tatsächlich angeschaut. Daraus haben sich folgende Ergebnisse ergeben:

Zuerst wurden die Leute gefragt, wie ihnen der Bot gefallen hat. Die Entscheidungen zwischen „gar nicht“(1) und „sehr!“(5) sehen wie folgt aus:

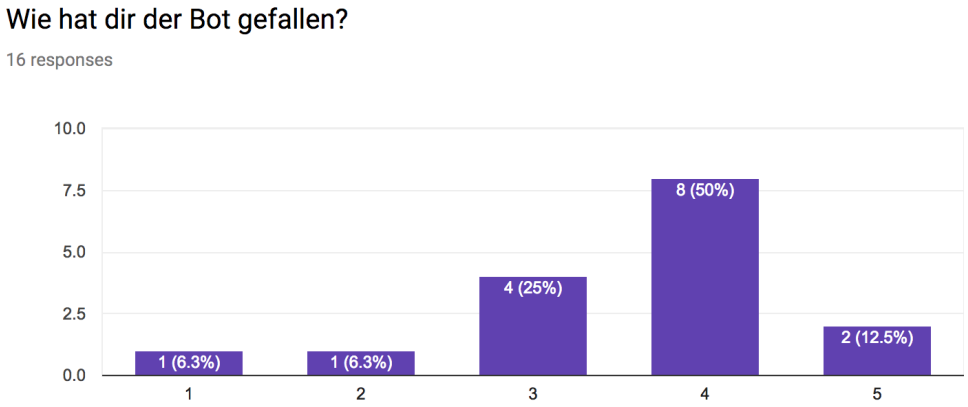


Abbildung 6.3: Antwort zu „Wie hat dir der Bot gefallen?“

Die meisten haben den Bot eher als gut befunden.

Als nächster Schritt wird die empfundene Funktionalität betrachtet, von Präferenzen, die von „absolut nicht“(1) bis „perfekt“(5) reichen:

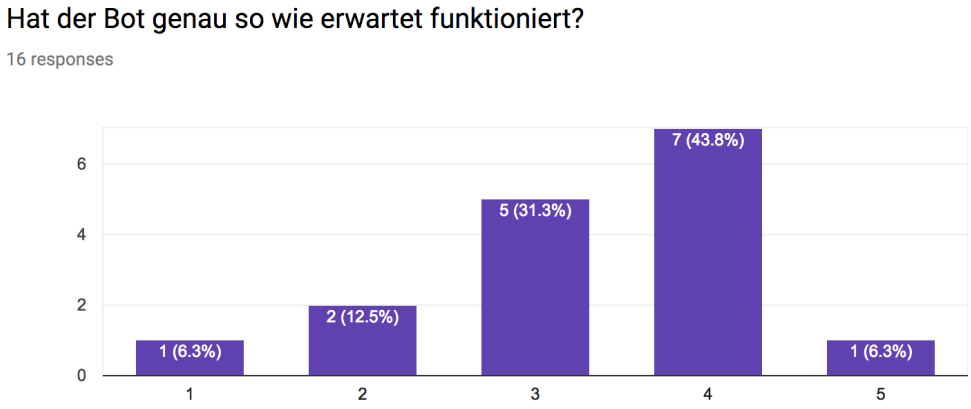


Abbildung 6.4: Antwort zu „Hat der Bot genau so wie erwartet funktioniert?“

Scheinbar gab es sowohl sehr enttäuschte Benutzer, die nichts Passendes gefunden haben, als auch solche, die das Bot für sehr funktionsfähig gehalten haben. Jedoch die meisten deuten auf die korrekte Funktionalität des Bots.

Auch wenn der Agent korrekt funktioniert, ist es wichtig zu wissen, ob die Leute sich eher an das „Internet“(1) oder an den „Bot“(5) wenden würden:

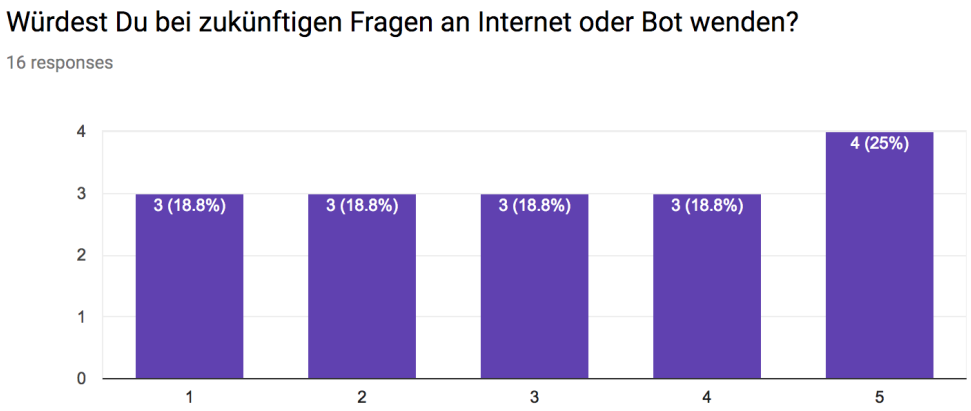


Abbildung 6.5: Antwort zu „Würdest Du bei zukünftigen Fragen an Internet oder Bot wenden?“

Dabei ist zu berücksichtigen, dass diese Präferenzen auch durch persönliche Gründe geprägt und damit nicht sehr objektiv sind. Es gibt viele Leute, die aufgrund ihrer Präferenz entweder einen Bot oder das Internet unabhängig von der Funktionalität immer bevorzugen würden.

Die nächste wichtige Frage ist, ob der Bot bei der Benutzung Spaß gemacht hat. Dabei variieren die Antworten zwischen „es war traurig“(1) und „sehr!“(5):

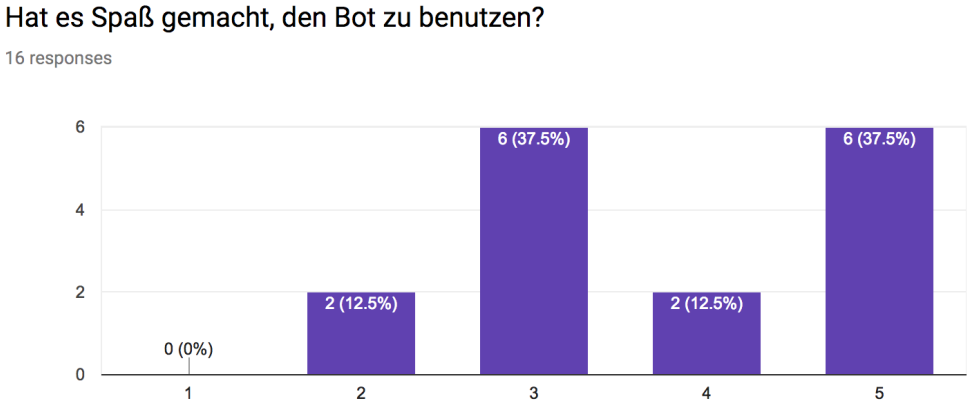


Abbildung 6.6: Antwort zu „Hat es Spaß gemacht, den Bot zu benutzen?“

Hier können wir sofort erkennen, dass der Spaßfaktor eher gelungen ist und durch weitere Implementierung und Verbesserungen ein noch besseres Ergebnis erzielt werden kann.

Weiterhin ist der Bot auch entwickelt worden, damit es den Anwendern ermöglicht wird, ihre Fragen auf einfache, benutzerfreundliche und aufwandsminimale Art und Weise beantwortet zu bekommen. Und somit ist es wichtig, ob es „schwierig“(1) oder „sehr einfach“(5) ist, den Bot zu benutzen:

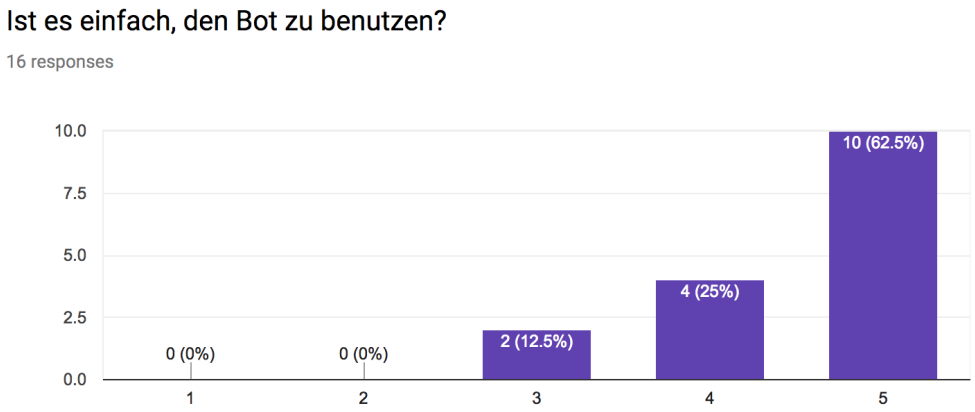


Abbildung 6.7: Antwort zu „Ist es einfach, den Bot zu benutzen?“

Diese Aufgabe der einfachen Bedienung wurde offenbar sehr gut erledigt, da die meisten Befragen die Anwendung als leicht empfunden haben. Ein weiterer wichtiger Faktor ist Präzision:

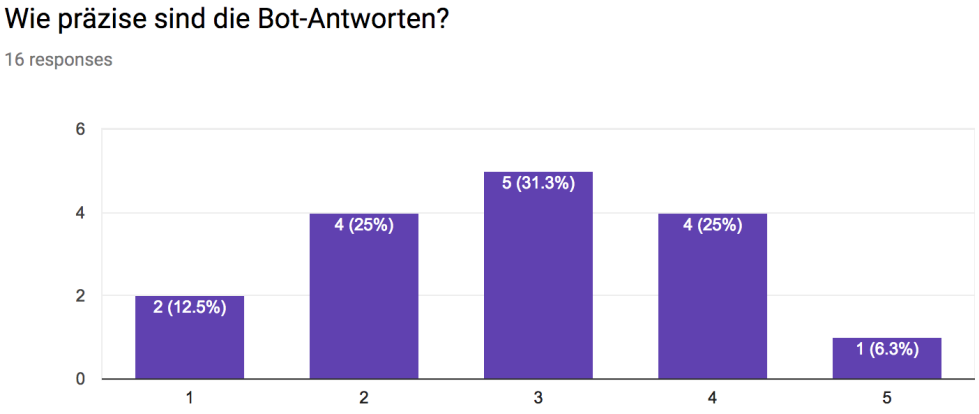


Abbildung 6.8: Antwort zu „Wie präzise sind die Bot–Antworten?“

Hier gibt es viele variierende Antworten zwischen „sehr ungenau“(1) und „sehr passend“(5). Das sollte aber nicht als besonders schlecht angesehen werden, da sich der Bot noch in der

Entwicklungsphase befindet. Er ist noch sehr jung und viele kleine Verbesserungen können die Funktionalität ständig verbessern.

Die folgende Frage stellt dar, wie viele der Befragten den Bot in der Zukunft „sehr selten“(1) oder „sehr oft“(5) benutzen würden:

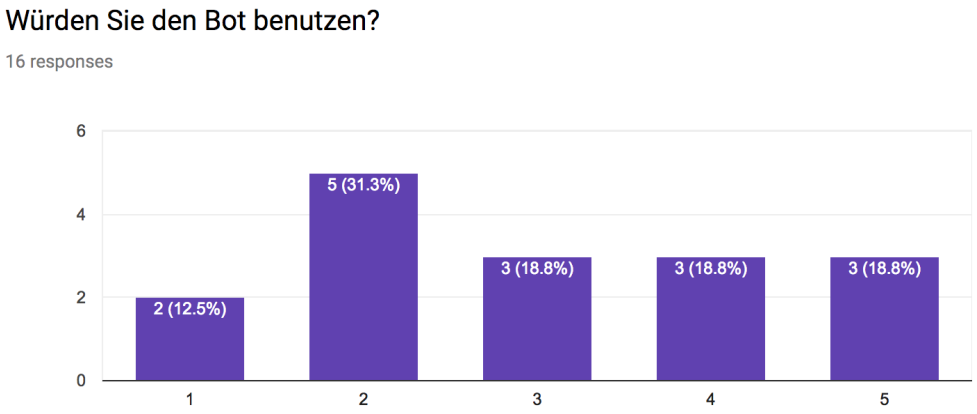


Abbildung 6.9: Antwort zu „Würden Sie den Bot benutzen?“

Hier sind die Antworten fast gleichverteilt und somit ein relativ gutes Ergebnis. Selbst in dieser Anfangsphase gibt es einige Leute, die sehr daran interessiert sind, den Bot zu benutzen.

Um herauszufinden, ob der Bot tatsächlich interessant ist, wurde nach Weiterempfehlungen gefragt. Dabei konnten die Befragten zwischen „eher nein“(1) und „eher ja“(5) wählen:

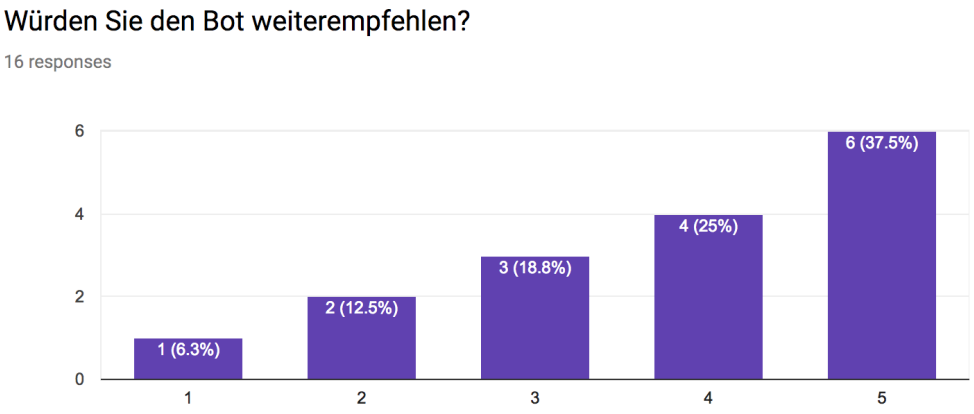


Abbildung 6.10: Antwort zu „Würden Sie den Bot weiterempfehlen?“

Diese Ergebnisse sind vielversprechend. Es ist wichtig, dass der Bot weiterempfohlen wird, denn daraus resultiert, dass er gebraucht wird. Damit diese Evaluation auch Input von den Benutzern bekommen konnte, wurde ein Textfeld angeboten:

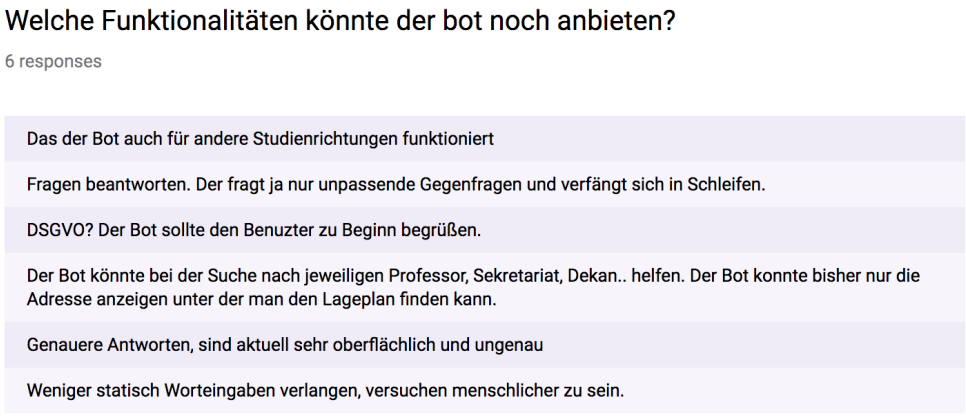


Abbildung 6.11: Antwort zu „Welche Funktionalitäten könnte der Bot noch anbieten?“

Die meisten Meinungen deuten darauf hinaus, dass die Funktionalität des Bots noch erweitert und natürlicher gestaltet werden sollte. Dieses Ziel ist natürlich sehr wichtig, dabei sollte aber auch nicht vergessen werden, dass die Antworten von einer künstlichen Intelligenz zurückgegeben werden und der Bot erst seit einigen Monate in der Entwicklung ist.

Als letztes sollte herausgefunden werden, welche Aktivitätsfelder bei dem Bot die wichtigsten sind:

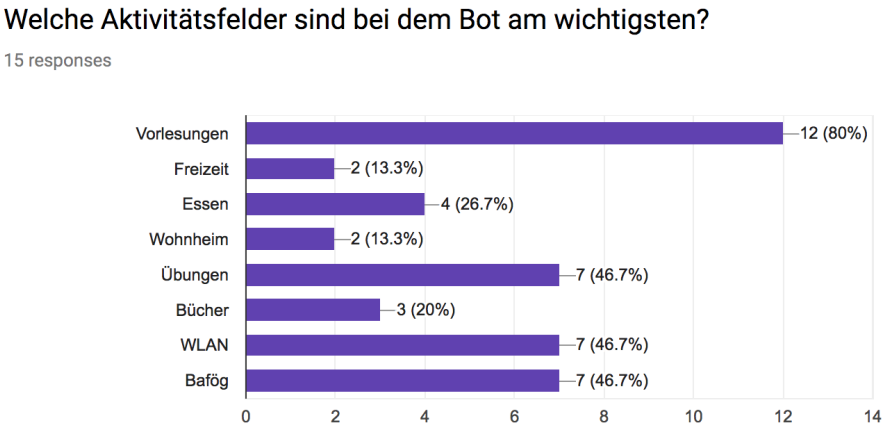


Abbildung 6.12: Antwort zu „Welche Aktivitätsfelder sind bei dem Bot am wichtigsten?“

Die meisten Interessenten haben das Thema „Vorlesung“ bevorzugt. Die Übungen, Internet und BAföG sind aber nicht weit entfernt. Selbst Themen wie Internet, Freizeit und Wohnungssuche wurden zum Teil berücksichtigt.

Die Evaluationsergebnisse ergeben eine klare Schlussfolgerung: Der Bot ist noch jung und bedarf vieler Verbesserungen, dennoch ist er heute schon interessant und erhält bereits erste mögliche Nutzer.

Kapitel 7

Fazit und Ausblick

Unser erzeugter Dialogflow–Agent für Studierende ist mittlerweile in der Lage mehrere Fragen zu beantworten und hat bereits viele Training–Beispiele erfahren. Der Bot hat von einigen Nutzern sogar sehr gute Feedbacks erhalten. Abschließend ist es aber auch wichtig, eine Aussicht zu erstellen, wie der Bot in der Zukunft möglicherweise verbessert und weiterentwickelt werden könnte.

Als erste Möglichkeit der Weiterentwicklung sollten detaillierte Antworten erwähnt werden. Das sind diese Antworten, die einer etwas längeren Konversation bedürfen. Beispielsweise würde der Bot bei einer Frage wie „Wie könnte mein Stundenplan aussehen?“ statt einer direkten Antwort aus Gitlab Snippet eine Gegenfrage stellen, wie „In welchem Semester bist du?“. Diese Zahl würde in dem „parameters“–Feld gespeichert und könnte dann im Python–Skript verwendet werden. Dies wurde in dem Kapitel über die Entwicklung des Bots bereits angesprochen. Somit könnte der Benutzer eine genauere Antwort erhalten. Diese Antwort würde einen möglichen Stundenplan, basierend auf dem aktuellen Semester, in dem sich der Studierende gerade befindet, enthalten.

Als zweiter Punkt ist zu nennen, dass die Tests in der Testing–Phase eventuell parallel ausgeführt werden könnten. Zurzeit ist dies, obwohl es trivial erscheint, nicht auf Knopfdruck umsetzbar und würde eine längere Recherche und eventuell eine neue Implementierung voraussetzen.

Drittens ist zu sagen, dass es effektiver wäre, wenn der Bot alle Fragen und Antworten auch aus Dokumenten, FAQs und mehreren Quellen eigenständig interpretieren könnte. Mit kleinen Schritten kann das Antwortproblem effektiver gelöst werden als mit dem Gitlab Snippet. Diese Vorgehensweise würde aber für die Umsetzung auch einige Zeit in Anspruch nehmen.

Zuletzt ist zu erwähnen, dass der Bot in die möglichen Chatplattformen oder auch in die Informatikwebseite der Heinrich–Heine–Universität eingebunden werden könnte. Dazu gibt es noch eine ergänzende Funktion, die den Bot vervollständigen würde. Die URL–Adressen sind derzeit (in Dialogflow–Bot GUI) nicht anklickbar. Das benutzte Dialogflow–Web–Demo–Chatfenster unterstützt nur Textformen, jedoch keine weiteren Kontexte. Daher wäre es ein wichtiger Weiterentwicklungsschritt herauszufinden, ob man den Bot tatsächlich eher auf der Webseite oder auf Chatplattformen zur Nutzung anbieten möchte. Falls die Wahl auf die Webseite fällt, sollte eine Integrität entwickelt werden, um die Links anklickbar zu machen.

Als Beispiel ist eine Lösung durch Kommunicate.io schon integriert. Dies wurde in der Entwicklung beschrieben. Die Seite ist unter die URL [1] erreichbar. Jedoch ist dies nicht die bestmögliche Lösung, da das Chatfenster einerseits Werbung von Kommunicate.io trägt und andererseits die Funktionalität eingeschränkt ist. Außerdem können in der Zukunft auch mögliche Kosten anfallen. Daher ist die Webseitenintegration immer noch ein sehr wichtiger Weiterentwicklungspunkt.

[1] „http://18.184.17.109:81/“

Literaturverzeichnis

[Ama18a] AMAZON WEB SERVICES: *Amazon EC2*. https://aws.amazon.com/de/ec2/. Version: Oktober 2018.

[Ama18b] AMAZON WEB SERVICES: *Amazon EC2 Instance IP Addressing*. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-instance-addressing.html. Version: Oktober 2018.

[Ama18c] AMAZON WEB SERVICES: *Amazon EC2-Instance-Typen*. https://aws.amazon.com/de/ec2/instance-types/. Version: Oktober 2018.

[Ama18d] AMAZON WEB SERVICES: *Amazon EC2 Security Groups for Linux Instances*. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html. Version: Oktober 2018.

[Ama18e] AMAZON WEB SERVICES: *Getting Started with IPv4 for Amazon VPC*. https://docs.aws.amazon.com/vpc/latest/userguide/getting-started-ipv4.html. Version: Oktober 2018.

[Ama18f] AMAZON WEB SERVICES: *Kostenloses Nutzungskontingent für AWS*. https://aws.amazon.com/de/free/. Version: Oktober 2018.

[Ama18g] AMAZON WEB SERVICES: *Step 1: Create Your EC2 Resources and Launch Your EC2 Instance*. https://docs.aws.amazon.com/efs/latest/ug/gs-step-one-create-ec2-resources.html. Version: Oktober 2018.

[AND18] ANDREA COLANGELO: *AWS Regions and Availability Zones: the simplest explanation you will ever find around*. https://cloudacademy.com/blo

g/aws-regions-and-availability-zones-the-simplest-explanation-you-will-ever-find-around/. Version: Oktober 2018.

[Arm18a] ARMIN RONACHER: *Flask is a microframework for Python based on Werkzeug, Jinja 2 and good intentions.* http://flask.pocoo.org/. Version: Oktober 2018.

[Arm18b] ARMIN RONACHER: *Quickstart*. http://flask.pocoo.org/docs/1.0/quickstart/#a-minimal-application. Version: Oktober 2018.

[art18] ARTEMGONCHARUK, GITHUB: *apiai-python-webhook*. https://github.com/xVir/apiai-python-webhook/blob/master/app.py. Version: Oktober 2018.

[Can18] CANONICAL LTD.: *systemd.service.5.gz*. http://manpages.ubuntu.com/manpages/cosmic/man5/systemd.service.5.html. Version: Oktober 2018.

[Dat18] DATA MONSTERS: *25 Chatbot Platforms: A Comparative Table*. https://chatbotsjournal.com/25-chatbot-platforms-a-comparative-table-aeefc932eaff. Version: Oktober 2018.

[Den18] DENISS STEPANOVS: *botium*. https://github.com/botium/botium. Version: Oktober 2018.

[Dia18a] DIALOGFLOW: *Configure fulfillment*. https://dialogflow.com/docs/fulfillment/configure. Version: Oktober 2018.

[Dia18b] DIALOGFLOW: *Developer Entities*. https://dialogflow.com/docs/reference/developer-entities. Version: Oktober 2018.

[Dia18c] DIALOGFLOW: *Entities overhkjhkjview*. https://dialogflow.com/docs/entities. Version: Oktober 2018.

[Dia18d] DIALOGFLOW: *How fulfillment works*. https://dialogflow.com/docs/fulfillment/webhook-slot-filling. Version: Oktober 2018.

[Dia18e] DIALOGFLOW: *How fulfillment works*. https://dialogflow.com/docs/fulfillment/how-it-works. Version: Oktober 2018.

[Dia18f] DIALOGFLOW: *Intents Overview*. https://dialogflow.com/docs/intents. Version: Oktober 2018.

[Dia18g] DIALOGFLOW: *Slot Filling*. https://dialogflow.com/docs/reference/slot-filling. Version: Oktober 2018.

[Dia18h] DIALOGFLOW: *System Entities*. https://dialogflow.com/docs/reference/system-entities. Version: Oktober 2018.

[Dia18i] DIALOGFLOW: *Training*. https://dialogflow.com/docs/training-analytics/training. Version: Oktober 2018.

[Flo18] FLORIAN TREML: *3 Steps: Setup Automated Testing for Google Assistant and Dialogflow*. https://chatbotsmagazine.com/3-steps-setup-automated-testing-for-google-assistant-and-dialogflow-de42937e57c6. Version: Oktober 2018.

[Fre18] FREE CSS: *MOTION FREE CSS TEMPLATE*. https://www.free-css.com/free-css-templates/page230/motion. Version: Oktober 2018.

[Gau18] GAUVAIN POCENTEK, MIKA MÄENPÄÄ: *Welcome to python-gitlab's documentation!* https://python-gitlab.readthedocs.io/en/stable/. Version: Oktober 2018.

[Git18] GITLAB DOCS: *Snippets*. https://docs.gitlab.com/ee/user/snippets.html. Version: Oktober 2018.

[Goo18] GOOGLE INC.: *Dialogflow: Python Client*. https://dialogflow-python-client-v2.readthedocs.io/en/latest/. Version: Oktober 2018.

[Har18] HARDIK MAKADIA, MARKETING EXECUTIVE AT MARUTI TECHLABS (2016-PRESENT): *Which are the best chatbot frameworks?* https://www.quora.com/Which-are-the-best-chatbot-frameworks. Version: Oktober 2018.

[Jac18] JACADA INC.: *Building Conversations with Wit.Ai*. https://jacada.zendesk.com/hc/en-us/articles/360007117251-Building-Conversations-with-Wit-Ai. Version: Oktober 2018.

[Jor18] JORDAN NOVET, CNBC: *Microsoft narrows Amazon's lead in cloud, but the gap remains large*. https://www.cnbc.com/2018/04/27/microsoft-gains-cloud-market-share-in-q1-but-aws-still-dominates.html. Version: Oktober 2018.

[Kom18] KOMMUNICATE.IO: *We, at Kommunicate, thrive to revolutionize the way your business interacts with customers.* https://www.kommunicate.io/about. Version: Oktober 2018.

[Mic18a] MICROSOFT: *Azure Bot Service – Preise*. https://azure.microsoft.com/de-de/pricing/details/bot-service/. Version: Oktober 2018.

[Mic18b] MICROSOFT: *What is cloud computing?* https://azure.microsoft.com/en-in/overview/what-is-cloud-computing/. Version: Oktober 2018.

[NEW18] NEWGENAPPS: *Top 5 Cloud Platforms and Solutions to Choose From*. https://www.newgenapps.com/blog/top-5-cloud-platforms-and-solutions-to-choose-from. Version: Oktober 2018.

[NGI18] NGINX INC.: *Welcome to NGINX Wiki!* https://www.nginx.com/resources/wiki/. Version: Oktober 2018.

[Red18] RED HAT: *HOW ANSIBLE WORKS*. https://www.ansible.com/overview/how-ansible-works. Version: Oktober 2018.

[SSH18] SSH COMMUNICATIONS SECURITY: *SSH COMMAND*. https://www.ssh.com/ssh/command/. Version: Oktober 2018.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 31. Oktober 2018 Lukas Miliunas

Diese CD enthält:

- eine *pdf*-Version der vorliegenden Bachelorarbeit
- die LaTeX- und Grafik-Quelldateien der vorliegenden Bachelorarbeit samt aller verwendeten Skripte
- die Quelldateien der im Rahmen der Bachelorarbeit erstellten Python–Software
- die Websites der verwendeten Internetquellen