Bachelorarbeit

Helper-Bot für E-Learning

Zürcher Hochschule für Angewandte Wissenschaft
School of Management and Law, Winterthur
Studiengang Wirtschaftsinformatik

Vorgelegt von:

Janick Michot

Matr.-Nr.: 16561276

Betreut von:

David Grünert

Abgabedatum:

23. Mai 2019

Management Summary

Wo früher Programmierkurse nur in das Curricula von technischen Studiengängen gehörten, ist heutzutage vermehrt auch bei Studiengängen ausserhalb der Informatik das Erlernen einer Programmiersprache gefordert. Auslöser dafür ist das zunehmende wirtschaftliche und gesellschaftliche Interesse an einer grundlegenden Programmierausbildung. Auch das Wirtschaftsdepartement der ZHAW (SML) hat dieses Interesse erkannt und bietet im Herbstsemester 2019 nicht-informatischen Studiengängen einen Programmier-Einführungskurs als Wahlpflichtmodul an. Nun verlangt eine solche Kurserweiterung zusätzliche Ressourcen für die Betreuung und Bewertung von Programmiernovizen bei der Lösung von Programmieraufgaben. Um gleichzeitig den Einsatz menschlicher Ressourcen zu reduzieren und um die Qualität eines Kurses zu verbessern, hielten in der Vergangenheit lernunterstützende Systeme Einzug in die akademische Programmierlehre.

Im Rahmen dieser Thesis wird erstmals ein solche Anwendung auf Basis der Chatbot-Technologe entworfen und erstellt. Mit einer prototypischen Umsetzung soll geprüft werden, ob sich die Idee eines Helper-Bots realisieren lässt, inwiefern qualitative Verbesserungen des Kurses hinsichtlich einer Reduktion der Antwortzeit möglich sind und ob eine solche Anwendung dabei hilft, menschliche Ressourcen in der Programmierlehre zu reduzieren.

Neben der Untersuchung der theoretischen Grundlagen und dem Forschungsstand gingen der Umsetzung des Helper-Bots insbesondere Überlegungen zur Anwendungsarchitektur voraus. Im Gegensatz zu herkömmlichen Chatbots beschränken sich die Funktionen des Helper-Bots nicht auf einfaches Question & Answer und stattdessen muss die Anwendung fähig sein, studentischen Code auszuführen und entsprechendes Feedback zu generieren. Die Anwendung besteht aus mehreren Teilsystemen, für die es allesamt passende Lösungen zu evaluieren galt. Aufgrund der Komplexität wurde für die Entwicklung ein inkrementelles Vorgehen gewählt.

Das Artefakt dieser Arbeit basiert auf der Chatbot-Plattform Dialogflow, welche aus den Benutzereingaben Absichten und Entitäten herausliest und diese an den Helper-Bot weitergibt. Zur statischen und dynamischen Code-Analyse wurde eine separate API erstellt, wo studentische Programme gesichert ausgeführt werden. Die Konfiguration des Helper-

Bots sowie die Formulierung von Hilfestellungen erfolgt wiederum über einen passwortgeschützten Admin-Bereich. Durch das Zusammenspiel dieser Komponenten wird es einem Studenten ermöglicht, unmittelbar und ohne Miteinbezug eines Dozenten Hilfe zur Lösung einer Programmieraufgabe zu erhalten.

Mit der Umsetzung des Helper-Bots wurde gezeigt, dass sich eine Chatbot-Anwendung zur Lernunterstützung aus technischer Sicht umsetzten lässt. Wenngleich der Helper-Bot eine qualitative Verbesserung hinsichtlich einer verkürzten Latenz von Hilfestellung ermöglich, kann der didaktische Nutzen erst mit dem Einsatz in der akademischen Programmierlehre bemessen werden. Gleiches gilt für die partielle Ersetzung von menschlichen Ressourcen. Schlussendlich hängt der Erfolg des Helper-Bots von der Akzeptanz der Studenten ab.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	=	Application Programming Interfaces
CSS	=	Cascading Style Sheets
HTML	=	Hypertext Markup Language
IDE	=	Integrated-Development-Environment
IO-Matching	=	Input-Output-Matching
JRE	=	Java Runtime Environment
KI	=	Künstliche Intelligenz
MVP	=	Minimum Viable Product
NLP	=	Natural Language Processing
NLU	=	Natural Language Understanding
NoSQL	=	Not only SQL
ORM	=	Object Relation Mapping
PaaS	=	Platform-as-a-service
POJO	=	Plain Old Java Object
REST	=	Representational State Transfer
RPC	=	Remote Procedure Call
SDK	=	Software Development Kit
SQL	=	Structured Query Language
UI	=	User Interface
URL	=	Uniform Resource Locator

1 Einleitung

Dieses einführende Kapitel erläutert die Ausgangslage, Relevanz und Problembeschreibung dieser Arbeit. Weiter wird basierend auf der Forschungsfrage die Zielsetzung, die thematische Abgrenzung sowie der inhaltliche Aufbau dargestellt.

1.1 Ausganslage und Relevanz

Aufbauend auf der Frage, warum die Programmierlehre zu einem relevanten wirtschaftlichen und gesellschaftlichen Thema geworden ist, wird in diesem Kapitel beschrieben, warum sich das Lernen und Lehren von Programmierkenntnissen aus didaktischer Sicht als kompliziert darstellt. Aus diesen Fragestellungen ergibt sich im nachfolgenden Kapitel die Problembeschreibung dieser Thesis.

1.1.1 Informatisches Denken als entscheidende Fähigkeit

Computational Thinking oder zu Deutsch Informatisches Denken beschreibt das menschliche Vermögen, reale Probleme algorithmisch zu erfassen sowie die Fähigkeit komplexe Probleme in lösbare Teilprobleme umzuformen (Jensen, 2017). Obschon die deutsche Übersetzung von Computational Thinking auf die Fähigkeiten eines Informatikers schliessen lässt, sieht Wing (2008) Informatisches Denken als entscheidende Fähigkeit in diversen Fachbereichen. So sorgt maschinelles Lernen beispielsweise für eine ganz neue Art und Weise, wie Statistiker denken und handeln (Wing, 2008). Nach vergleichbarem Prinzip verändert eine algorithmische Spieltheorie wie Ökonomen denken, Nanocomputing wie Chemiker denken und Quantum Computing wie Physiker denken (Wing, 2008).

Wing (2008) und Jensen (2017) appellieren daher an Schulen, Hochschulen und Weiterbildungen, diese Denkweise zu fördern. Auf schulischer Ebene muss ein Pflichtfach Informatik eingeführt werden und auf akademischer Ebene gehören Einführungskurse zu IT-Themen in den Curricula von nicht informatischen Studiengängen (Strickroth & Pinkwart, 2017). In diesen Kursen spielt die Programmierung typischerweise eine zentrale Rolle (Strickroth & Pinkwart, 2017). Zwar ist es oft nicht das Ziel, eine vollumfängliche Ausbildung in der Programmierung zu ermöglichen, dennoch sollen Programmiersprachen den Studierenden als Werkzeuge greifbar gemacht werden (Jensen, 2017).

1.1.2 Zunehmende Teilnehmerzahlen an Programmierkurse der SML

Auch die School of Management and Law (SML), das Wirtschaftsdepartement der Zürcher Hochschule für Angewandte Wissenschaften (ZHAW), ist sich der zunehmenden Relevanz bewusst. Ab dem Herbstsemester 2019 wird ein Programmier-Einführungskurs sowohl dem Studiengang General Management als Wahlpflichtmodul, als auch internen Mitarbeitern und Masterstudenten als Konvergenz-Veranstaltung angeboten. In diesem Kurs werden Grundlagen zur systematischen und objektorientierten Programmierung vermittelt und die algorithmische und objektorientiere Denkweise, das informatische Denken, geschult.

Eine solche Kurserweiterung für Programmiernovizen dürfte sich als kompliziert erweisen, da gemäss Bauer-Messmer, Fässler und Wyss (2007) fast jede Person, die noch nie mit der Programmierung zu tun hatte, diese zumeist als hochkompliziert und sehr schwer erlernbar empfindet. So erwirbt entsprechend einer Studie von McCracken, et al. (2001) ein beträchtlicher Anteil der Studierenden während dem Programmier-Einführungskurs nicht einmal ein grundlegendes Niveau an Programmier-Fähigkeiten und gemäss Bennedsen und Caspersen (2007) bestehen viele Studierende ihren Einführungskurs gar nicht erst. Nachfolgend werden einige Gründe dafür aufgezählt, weshalb Studenten das Erlernen von Programmierfähigkeiten schwerfällt und weshalb Programmierkurse aus didaktischer Sicht eine Herausforderung darstellen.

1.1.3 Didaktische Herausforderungen

Viele Studierende können zwar leicht Konzepte und Regeln aufnehmen und wiedergeben, was beim Programmieren jedoch nicht genügt. Hierbei sind Wissen und Fertigkeiten notwendig, wie man das Gelernte in der Praxis anwendet (Bauer-Messmer et al., 2007). Zu diesem Problem trägt die passive Schülerrolle im Frontalunterricht und die Aufteilung zwischen Theorie (Unterricht) und Praxis (Übungen) bei (Bauer-Messmer et al., 2007). Schüler konsumieren Wissen, sind aber mit dessen Anwendung oft überfordert und stossen beim Erstellen, Testen oder Debuggen von Code schnell an ihre Grenzen. Neben den Herausforderungen, strukturierte Problemlösungen zu erlernen und zu verstehen, wie Programme ausgeführt werden, gehört eine starre Syntax mit teilweise verwirrenden Namen zum Lernprozess (Kelleher & Pausch, 2005). Diese Herausforderungen gleichzeitig zu bewältigen, überfordert viele Lernenden und wirkt sich oft sogar entmutigend aus (Kelleher & Pausch, 2005).

Weiter müssen Programmierkurse sowohl Studenten mit Vorwissen fordern, als auch Studenten ohne Vorwissen motivieren und fördern (Bauer-Messmer et al., 2007). Hierbei zeigt sich auf universitärerem Niveau das Problem der unterschiedlichen schulischen Hintergründe der Studierenden und den daraus resultierenden Wissensdifferenzen. Es ist fast unmöglich einen Unterricht zu gestalten, welcher alle Studenten fordert, ohne viele zu überfordern (Bauer-Messmer et al., 2007). Die Lehrenden versuchen den Lernprozess zwar bestmöglich zu unterstützen, jedoch bleibt ihnen oft nicht genügend Zeit, um auf individuelle Probleme und Bedürfnisse jedes Lernenden einzugehen (Strickroth & Pinkwart, 2017).

Fehlende Zeit als limitierender Faktor zeigt sich vor allem bei der Betreuung und Bewertung studentischer Abgaben. Eine einfache binäre Aussage über den Programmcode trägt nicht viel zum Lernerfolg bei und für eine ausführliche Beurteilung bleibt zumeist kaum Zeit (Nguyen et al., 2014). Dies obschon regelmässiges und ausführliches Feedback als wichtiger Aspekt des erfolgreichen Lernens gilt (Nguyen et al., 2014). Damit ein Student seine Programmierkenntnisse verbessern kann, muss ihm bewusst sein, was seine Lösung für eine Qualität aufweist und inwiefern diese von den gängigen Standards abweicht (Boud und Molloy, 2013). Zudem ist eine konstruktive Bewertung durch eine Lehrperson mit einer Latenz zwischen Abgabe und Feedback verbunden.

1.2 Problemstellung

Um die Qualität eines Programmierkurses zu verbessern und um den Einsatz menschlicher Ressourcen zu reduzieren, fanden in der Vergangenheit vermehrt lernunterstützende Tools Anwendung in der akademischen Programmierlehre (Strickroth & Pinkwart, 2017). Computer sind in der Lage, studentische Programme automatisch zu analysieren und das Ergebnis der Bewertung als Feedback an Studenten zurückzugeben (Strickroth & Pinkwart, 2017). Einen wichtigen Aspekt stellt dabei der Umgang mit fehlerhaften oder falschen Lösungen dar, die auf unterschiedlichen oder fehlerbehafteten Lösungsstrategien beruhen (Strickroth & Pinkwart, 2017). Herkömmliche Programmierumgebungen, sogenannte Integrated-Development-Environments (IDE), sind diesen Herausforderungen gemäss Pears et al., (2007) zumeist nicht gewachsen. Gründe dafür sind einerseits die Funktionen, welche für Novizen oft zu kompliziert sind, und andererseits sind die in den IDEs eingebauten Codeanalysen und Rückmeldungen nicht auf Programmiernovizen zugeschnitten, sodass sie eher verwirren als helfen (Pears et al., 2007).

Ein solches IDE wird auch im Programmierkurs an der SML eingesetzt. Obschon das IDE Möglichkeiten zur automatischen Bewertung besitzt, reichen die Funktionalitäten nicht zur automatisierten Betreuung von Programmiernovizen aus. Stattdessen erfolgen repetitive Hilfestellungen weiterhin durch einen Dozenten und ein Student wird beim Auftreten eines Problems während der Bearbeitung von Programmieraufgaben unterbrochen.

1.3 Forschungsfragen

Ziel dieser Forschung ist es den Nutzen der Chatbot-Technologie in der Programmierlehre zu evaluieren. Zu diesem Zweck soll eine Chatbot-Anwendung konzipiert und umgesetzt werden, welche einem Studenten bei Bedarf Hilfestellungen zur Lösung einer Programmieraufgabe gibt. Der Helper-Bot soll auf dem existierenden E-Learning-Angebot aufbauen und dort eingesetzt werden, wo die Hilfeleistung durch das IDE an seine Grenzen stösst. Die Forschungsfrage dieser Thesis lautet dementsprechend:

> *Wie kann eine Chatbot-Anwendung in die existierende E-Learning-Umgebung implementiert werden, sodass einem Studenten bei Bedarf umgehend und ohne Miteinbezug eines Dozenten Hilfe geleistet werden kann?*

Die Forschungsfrage lässt sich nach dem Prinzip des Aufteilens eines umfassenden Problems in Teilprobleme in mehrere Teilfragen zerlegen:

1. Wie gestaltet sich der aktuelle Forschungsstand lernunterstützender Software und welchen Nutzen bringen diese Systeme für Lernende und Lehrende?
2. Welche Plattformen zur Erstellung eines Chatbots gibt es und welche eignet sich im Kontext dieser Arbeit am besten?
3. Wie kann studentischer Programmiercode statisch wie auch dynamisch überprüft und didaktisch sinnvolles Feedback generiert werden?
4. Wie lässt sich das Artefakt in die bestehende Umgebung implementieren und welche Systemarchitektur eignet sich dazu?
5. Welche Konfiguration- und Administrationsmöglichkeiten muss das System zur Definition von Tests und Formulierung von Feedback bieten?

1.4 Abgrenzung

In dieser Arbeit wird kein eigener Chatbot (keine eigene NLU-Komponente, vgl. dazu Kapitel 3.1.1) entwickelt, sondern eine passende Plattform evaluiert und in eine Anwendung implementiert. Die Kursmanagementplattform Moodle und repl.it als Programmierumgebung (IDE) existierend bereits und müssen gegebenenfalls konfiguriert werden. Ebenfalls ist die Formulierung von Programmieraufgaben und die Verwaltung von Kursseiten nicht Teil dieser Arbeit. Obgleich einige Tests zur Evaluation der Anwendung geschrieben werden, ist es nicht das Ziel dieser Arbeit, Aufgaben und Tests zu konzipieren.

2 Vorgehen und Methoden

Als Beschreibungen des Vorgehens und der Methoden zeigt dieses Kapitel die wissenschaftliche Herangehensweise dieser Arbeit. Das Kapitel umfasst sowohl die methodische Beschreibung der vorgängigen Analyse der Ausgangslage, wie auch die eigentliche Umsetzung des Helper-Bots. Zudem wird in einem dritten Teil der Aufbau dieser Arbeit erörtert.

2.1 Analyse der Ausgangslage

Der eigentlichen Umsetzung des Helper-Bots voraus geht eine Analyse der Ausgangslage. Diese umfasst zunächst eine theoretische Untersuchung der Chatbot-Technologie sowie der automatischen Code-Bewertung. Hierzu werden die wichtigsten Terminologien untersucht und in den Kontext dieser Arbeit gestellt.

Darauf aufbauend, respektive parallel zur theoretischen Analyse wird der Forschungsstand mittels systematischem Literaturreview untersucht und die für diese Arbeit relevanten Ansätze zur automatisierten Bewertung sowie entsprechende Systeme eruiert. Hierbei lieget der Fokus noch nicht konkret auf der Chatbot-Technologie, sondern allgemein auf lernunterstützenden Systemen. Gleichwohl soll die Darstellung des Forschungsstands dazu dienen, Nutzen und Möglichkeiten des Helper-Bots darzulegen und relevante Ansätze für die Umsetzung zu identifizieren. Aus der Auswertung des Forschungsstands ergeben sich sodann Anforderungen an den Helper-Bot.

Der Definition von Anforderungen geht jedoch die Analyse des fachlichen Kontextes voraus, welche als dritter Aspekt der Ausgangslage ebenso zentral für diese Arbeit ist. Darin wird das bestehende E-Learning-Angebot des Kurses Software Engineering an der SML analysiert. Das existierende soziotechnische System wird dokumentiert und überprüft, inwiefern sich dieses um einen Chatbot erweitern lässt.

Auf Basis des identifizierten Forschungsstands und entsprechend des gegebenen Kontexts werden anschliessend die Anforderungen an den Helper-Bot definiert und damit die Forschungsfragen konkretisiert. Aufgrund des Zusammenspiels mehrerer Teilsysteme, die als Summe das Artefakt dieser Arbeit ausmachen, gestaltet sich die Anforderungsdefinition als kompliziert. Um dieser Komplexität entgegenzuwirken wird das Gesamtsystem zur Anforderungsdefinition in einzelne Komponenten zerlegt. Mit der Zerlegung des

Systems in Teilsysteme werden die Anforderungen an den Helper-Bot an die einzelnen Komponenten weitergegeben.

Infolge dessen, dass die Wahl einer Chatbot-Plattform der Umsetzung vorausgeht, erfolgt diese direkt nach der Anforderungsdefinition. Verschiedene Plattformen werden untersucht und eine für den Helper-Bot passende evaluiert. Hierzu findet ein Vergleich der einzelnen Plattformen nach zuvor definierten Bewertungskriterien statt.

2.2 Inkrementelle Umsetzung

Aufgrund der Komplexität wurde für die Umsetzung des Chatbots ein inkrementelles Vorgehen gewählt, welches sich zudem an der Komponenten-Zerlegung orientiert. Beim inkrementalen Vorgehen nähert man sich in mehreren aufeinanderfolgenden Iterationen schrittweise der endgültigen Lösung (Wieczorrek & Mertens, 2008, S. 55ff.). Eine Iteration ist dabei eine zeitlich und fachlich abgeschlossene Einheit, welche jeweils sämtliche Projektphasen beinhalten. Aus jeder Iteration resultiert ein funktionierendes und einsetzbares Zwischenprodukt, welchem inkrementell neue Funktionalitäten hinzugefügt werden (Wieczorrek & Mertens, 2008, S. 55ff.). Der Umfang neuer Funktionen wird jeweils in Form sogenannter Releases definiert und erweitert.

Ausgehend von der Wahl einer Chatbot-Plattform, müssen in den ersten Iterationen Grundzüge des Helper-Bots umgesetzt werden. Die darin gewählte Architektur muss einen schrittweisen Ausbau des Systems erlauben und Schnittstellen müssen frühzeitig festgelegt werden (Wieczorrek & Mertens, 2008, S. 55ff.). Nur so ist ein inkrementeller Aufbau und die Integration sämtlicher Komponenten über mehrere Iterationen möglich. In jedem Release muss konkretisiert werden, was umgesetzt wird, welche Komponenten davon abhängen, inwiefern diese interdependent sind und welche Tools sich für die Lösungsumsetzung eignen. Dadurch ergibt sich die finale Lösung und deren Systemarchitektur erst nach mehreren Iterationen und auf Basis von Zwischenergebnissen.

2.2.1 Minimal Viable Product

Die durch das inkrementelle Vorgehen entstehenden Zwischenergebnisse orientieren sich zudem am Konzept des Minimal Viable Products und dem Ziel, schon zu einem frühen Stadium einsetzbare Zwischenergebnisse zu erhalten. Ohnehin wird in dieser Arbeit kein praxisreifes Produkt erstellt, sondern eine prototypische Umsetzung davon. Das Konzept

des Minimal Viable Products unterscheiden sich dahingehend vom bisherigen Modell getriebenen Web Engineering, dass viel früher mit einer Visualisierung der Applikation begonnen wird (Rivero et al., 2012, S. 152). Anstatt eines Top-Down Ansatzes, bei welchem zuerst das Back-End, dann die Struktur (HTML) und zuletzt das User Interface (UI) entwickelt werden, sollen diese Bestandteile parallel und iterativ realisiert werden (Rivero et al., 2012, S. 152).

2.2.2 Testumgebung

Für die prototypische Umsetzung wird hierzu die cloudbasierte Plattform Heroku verwendet. Heroku funktioniert als Platform-as-a-service (PaaS) und hilft dabei Anwendungen zu erstellen, überwachen und skalieren, ohne dass dafür fundierte Kenntnisse von System-Administration notwendig sind (Heroku, o. J.). Dadurch kann sich ein Entwickler auf die Realisierung einer Anwendung konzentrieren und muss sich nicht um die Server-Infrastruktur kümmern.

2.3 Aufbau der Arbeit

Die Untersuchungen zur grundlegenden Theorie, dem Forschungsstandes und dem fachlichen Kontext dienen als Ausgangslage dieser Arbeit und befinden sich im anschliessenden Kapitel 3. Gleichzeitig beantwortet dieses Kapitel die erste Forschungsfrage sowie Ansätze der dritten und vierten. Ausgehend von den Forschungsfragen und den Erkenntnissen aus der Ausgangslage werden im Kapitel 4 Anforderungen an den Helper-Bot abgeleitet. Die Evaluation einer Plattform zur Erstellung eines Chatbots findet direkt nach der Anforderungsdefinition im separaten Kapitel 5 statt. Damit wird zugleich die zweite Forschungsfrage beantwortet.

Die Darstellung der Ergebnisse obliegt den beiden Kapiteln 6 und 7, welche die Entwicklung der Systemarchitektur über die einzelnen Iterationen, respektive die aus der Arbeit resultierende Anwendung und ihre Komponenten zeigen. Darin werden die in Kapitel 4 definierten Anforderungen und deren Umsetzung behandelt. Abgerundet wird diese Arbeit durch Kapitel 8, in welchem der Helper-Bot evaluiert und die Anforderungsumsetzung diskutiert wird. Ebenfalls wird darin die Beantwortung der Forschungsfragen kritisch gewürdigt wird und eine Schlussfolgerung gezogen.

3 Analyse

Zunächst werden in diesem Kapitel theoretische Grundlagen erläutert, die im Kontext dieser Arbeit relevant sind. Ebenfalls zur Untersuchung der Ausgangslage gehört ein Überblick über den Forschungsstand, in welchem verschiedene für diese Thesis relevanten Ansätze der automatisieren Lernunterstützung sowie entsprechende Systeme präsentiert werden. Neben der theoretischen Auseinandersetzung mit den Grundlagen und dem Forschungsstandes, befasst sich dieses Kapitel zudem mit dem fachlichen Kontext dieser Arbeit.

3.1 Theoretische Grundlagen

Der Fokus dieses Kapitels liegt nicht auf einer fundierten, theoretischen Auseinandersetzung mit der Materie, sondern soll dem Leser einen anwendungsbezogenen Überblick über die wichtigsten Begrifflichkeiten dieser Arbeit geben. Hierzu zählen Begriffe im Kontext der Chatbot-Technologie sowie Terminologien im Zusammenhang mit der Code-Bewertung und der Systemarchitektur. Das Kapitel kann entweder vorweg gelesen werden oder bei Bedarf konsultiert werden. In der Arbeit wird jeweils auf das entsprechende theoretische Kapitel verwiesen.

3.1.1 Chatbot

Mit den Begriffen Chatbots, Chatterbots oder kurz Bots bezeichnet man textbasierte Dialogsysteme, welches im Rahmen einer Mensch-Maschinen-Kommunikation zum Einsatz kommen (Eicher, 2016). So viel zur allgemeinen Definition von Chatbots. Aus Sicht von Shevat (2017, S. 13) sind Chatbots lediglich eine neuartige Möglichkeit, Services über ein konversationsbasiertes Interface anzubieten. So sind Chatbots nur das Interface zu einem Service und nicht der Service selbst, vergleichbar mit einer Webseite (Shevat, 2017, S. 13). Chatbots halten dabei an der Art und Weise fest, wie Menschen kommunizieren und versuchen, sich an unsere Denkweise anzupassen (Shevat, 2017, S. 64).

Fälschlicherweise wird künstliche Intelligenz (KI) oft als Essenz von Chatbots verstanden, obschon ein Chatbot grundsätzlich kein KI benötig (Shevat, 2017, S. 95). KI ist dabei nicht eine einzelne Sache, sondern eine Reihe von Tools, mit welchen ein Chatbot ausgerüstet werden kann (Shevat, 2017, S. 95). Im Rahmen dieser Arbeit und der Evaluation

einer geeigneten Plattform sind zwei Anwendungen von KI relevant: Natural Language Understanding (NLU) (Kapitel 3.1.1.1) und Conversation Management (Kapitel 3.1.1.5).

3.1.1.1 Natural Language Understanding

Natural Language Understanding (NLU) als Teilgebiet von Natural Language Processing (NLP) befasst sich mit der Verarbeitung unstrukturierter Eingaben in natürlichsprachige Form und deren Umwandlung in eine strukturierte und für Maschinen verständliche Form (Liddy, 2001). Unabhängig der Sprache gelten sämtliche gesprochenen oder schriftlichen Benutzereingaben als natürlichsprachig mit der Voraussetzung, dass die Sprache auch tatsächlich zur Kommunikation zwischen Menschen verwendet wird (Liddy, 2001). Im Gegenzug zu NLP befasst sich NLU noch konkreter mit dem Ziehen von Schlussfolgerungen aus einer Benutzereingabe (Liddy, 2001). Grundsätzlich ermöglicht NLU dadurch die Kommunikation zwischen Mensch und Maschine in menschlicher Sprache.

3.1.1.2 NLU-Plattformen

Der Begriff NLU-Plattformen umfasst gemäss Canonico und De Russis (2018) sowohl Plattformen, mit welchen Fähigkeiten, respektive Skills für Gesprächsastenten (Siri, Google Assistant, Alexa usw.) erstellt werden können, als auch Plattformen, über welche ein eigener Chatbot realisiert werden kann. Im Fall dieser Arbeit wird eine NLU-Plattform dazu benötigt, einen eigenen Chatbot zu entwickeln. Dazu gibt es verschiedene cloudbasierte Lösungen, die durch Parametrisierung und Training so konfiguriert werden können, dass sie zu bestimmten Benutzereingaben entsprechenden reagieren (Canonico & De Russis, 2018). Die bekanntesten dieser NLU-Plattformen werden im Kapitel 5 untersucht und eine passende evaluiert. Wichtig in Bezug auf diese Plattformen ist das Konzept der Intents und Entities. Diese beiden Begriffe werden nachfolgend beschrieben.

3.1.1.3 Intents

Ein Intent oder zu Deutsch eine Absicht repräsentiert die Zuordnung zwischen dem, was der Benutzer will und derjenigen Aktion, welche der Chatbot ausführen soll (Canonico & De Russis, 2018). Möglich macht diese Zuordnung NLU, die Verarbeitung von grossen Datenbeständen früher Gespräche (Trainingsset) sowie die Parametrisierung des Chatbots (Canonico & De Russis, 2018). Dadurch lassen sich Benutzereingaben wie «Ich möchte ein Filmticket kaufen» oder «Lasst uns ein Ticket für den Filme kaufen» zur entsprechenden Absicht, ein Ticket zu kaufen, zuordnen (Shevat, 2017).

3.1.1.4 Entities

Eine Entität ist hingegen ein Parameter, der aus der natürlichen Spracheingabe extrahiert wird (Canonico & De Russis, 2018). Alle wichtigen Daten, die man aus der Anfrage eines Benutzers erhalten möchten, haben eine entsprechende Entität und werden als Kontextvariablen bezeichnet (Shevat, 2017). Bei der Benutzereingabe «Wie ist das Wetter in Paris?» und der dazugehörigen Intention «nach dem Wetter fragen» wäre `Paris` eine mögliche Entität namens `city`, welche jede Stadt umfassen könnte (Canonico & De Russis, 2018). NLU-Plattformen erlauben es, sowohl neue Intents und Entitäten zu erfassen, als auch existierende (vordefinierte) wiederzuverwenden (Canonico & De Russis, 2018).

3.1.1.5 Conversation Management

Die Extraktion von Entitäten und Absichten ist ein wichtiger, aber grundlegender Aspekt eines Chatbots (Shevat, 2017, S. 96). Es gibt noch eine weitere Ebene des Gesprächsmanagements, welche sich mit dem Verständnis des Kontextes und dem Wechseln zwischen Subkonversationen befasst (Shevat, 2017, S. 96). Menschen können leicht mit einem Gesprächswechsel umgehen, für Maschinen ist dies jedoch eine schwierige Angelegenheit (Shevat, 2017, S. 96). So kann die gleiche Absicht unterschiedlich formuliert sein oder die gleiche Entität je nach Kontext unterschiedliche Zustände haben. Um trotzdem jeweils die passende Antwort zum erkannten Intent zu finden und um Gesprächsverläufe zu erzeugen, wird Conversation Management benötigt (Shevat, 2017, S. 96). Obwohl es auch regelbasiertes Conversation Management gibt, zeigen sich bei komplexeren Systemen erhebliche Vorteil von KI (Shevat, 2017, S. 96). Dazu gehört auch das Hinzulernen bei neuen Formulierungen einer Absicht oder einer Entität (Shevat, 2017, S. 96).

3.1.2 Automatische Bewertung

Mit automatischer Bewertung ist in dieser Arbeit jede Art von Analyse und Generierung von Feedback gemeint, die automatisch durch ein System erfolgt. Nachfolgend wird die Unterscheidung zwischen statischer und dynamischer Codeanalyse sowie unterschiedliche Bewertungsmetriken und Feedbackansätze erörtert.

3.1.2.1 Statische Codeüberprüfung

Statische Codeanalyse findet auf der Ebene des Quellcodes statt, wobei der Code nicht ausgeführt werden muss (Strickroth & Pinkwart, 2017). Hauptsächlich wird damit die syntaktische Korrektheit überprüft (Strickroth & Pinkwart, 2017). Weiter kann mittels

statischer Codeüberprüfung der Coding-Style (ungenutzte Variablen, Einhaltung von Style-Regeln usw.), Softwaremetriken (Anzahl Attribute, Operationen oder Codezeilen) und das Design (Vergleich mit einer Musterlösung) untersucht werden (Ala-Mutka, 2005). Obschon des breiten Spektrums an Überprüfungsmöglichkeiten liegt der Fokus dieser Arbeit auf der syntaktischen Korrektheit eines Programms.

3.1.2.2 Dynamische Codeanalyse

Bei der dynamischen Codeanalyse muss der Code hingegen ausgeführt werden, weshalb diese Art des Assessment grundsätzlich in einer gesicherten Umgebung stattfinden sollte (Strickroth & Pinkwart, 2017). Obschon mit der dynamischen Codeanalyse entsprechend Ala-Mutka (2005) mehrere Aspekte überprüft werden können, konzentriert sich diese Arbeit auf die dynamische Überprüfung der Funktionalität. Dabei findet die Überprüfung des Programmes auf Basis vorher definierter Testeingaben und der entsprechenden Ausgaben statt (Ala-Mutka, 2005). Ein solcher Test kann entweder die Ausgabe des gesamten Programms oder einer einzelnen Methode bewerten. Ala-Mutka (2005) unterscheidet dabei zwischen Blackbox-Tests für die gesamte Überprüfung und Whitebox-Tests für die methodenspezifische Überprüfung. In dieser Arbeit werden die beiden Terminologien Input/Output-Matching oder kurz IO-Matching zur Überprüfung des gesamten Codes und Unit-Tests zur Überprüfung einzelner Methoden verwendet.

3.1.2.3 Bewertungsmetriken

Caiza und Álamo Ramiro (2013) unterscheiden zwischen den in Tabelle 1 dargestellten Bewertungsmetriken. Darin sind mehrere der zuvor beschriebenen Tests enthalten. Von Relevanz für dieser Arbeit sind die beiden Kategorien Ausführung und Funktionstests mit den Metriken Kompilierung, Ausführung und Funktionalität.

Tabelle 1: Bewertungsmetriken (Caiza & Álamo Ramiro, 2013)

Kategorisierung		Metrik
Ausführung		Kompilierung (z.B. Compiler)
Funktionstests		Ausführung (z.B. Interpreter)
		Funktionalität (System- oder Methodenlevel)
Nicht funktionale Tests	Spezielle Anforderungen	Spezielle Anforderungen an eine Aufgabe
	Wartbarkeit	Design
		Stil
		Komplexität
	Effizienz	Ausführungszeit
		Nutzung physischer Ressourcen

3.1.3 Systemarchitektur

Als dritter Theorie-Teil befasst sich dieser Abschnitt mit den wichtigsten Begrifflichkeiten rund um die Schnittstellen und den Datenaustausch zwischen den einzelnen Teilsystemen. Dieses Kapitel ist hinsichtlich der Anforderungsdefinition in Kapitel 4, der Architekturdokumentation in Kapitel 6 und der Resultate in Kapitel 7 wichtig.

3.1.3.1 iFrame

Ein iFrame (kurz für inline-Frame) ist ein HTML-Element, über welches eine externe Webseite eingebettet werden kann (Christensson, 2015). Dieses kann überall innerhalb eines Webseitenlayouts eingefügt werden (Christensson, 2015).

3.1.3.2 Application Programmng Interface (API)

Ein Paradigma der Programmierung in der heutigen Zeit ist, dass Software kaum von Grund auf selber geschrieben wird und stattdessen vorhandene Services, Frameworks und Libraries beigezogen (Hunter II, 2017). Application Programming Interfaces (API) nehmen bei diesem Wiederverwendungsansatz eine immer wichtigere Rolle ein (Hunter II, 2017). Mit den primitiven CRUD-Operationen (Create, Read, Update und Delete) können Daten per http-Request an einen Service gesendet werden und man erhält die prozessierten Ergebnisse zurück (Hunter II, 2017). Hierbei unterscheidet sich eine API von einem Webhook, welcher eventbasiert ausgelöst wird und nicht aktiv aufgerufen werden muss.

3.1.3.3 Webhook

Ein Webhook ist eine Ereignisbenachrichtigung, die Daten per HTTP und typischerweise als POST-Anfrage an eine bestimmte URL überträgt (repl.it, o. J.). Die URL definiert dabei eine Ressource, welche die Daten der POST-Anfrage prozessiert. Ein Webhook wird also immer beim Auftreten eines bestimmten Ereignisses ausgelöst. Im Kontext dieser Arbeit beispielsweise dann, wenn der Student eine Lösung abgibt.

3.1.3.4 Software Development Kit (SDK)

Ein Software Development Kit (SDK) bietet ein Set an Tools mit Bibliotheken, relevanter Dokumentation, Code-Beispielen, Prozessen und/oder Anleitungen, die es Entwicklern ermöglichen, Softwareanwendungen für eine bestimmte Plattform zu erstellen (Sandoval, 2016). APIs sind oftmals Bestandteil eines SDK (Sandoval, 2016).

3.2 Stand der Forschung

Die nachfolgende Untersuchung von Tools zur akademischen Lernunterstützung stützt sich auf derjenigen von Strickroth und Pinkwart (2017) ab, welche unterschiedliche Sichtweitweisen und Klassifikationsansätze in einem Literaturreview analysiert haben. Neben zahlreichen publizierten Systemen, dürfte die Anzahl nicht publizierter Systeme, die in der universitären Praxis Anwendung finden, noch weitaus höher sein (Strickroth & Pinkwart, 2017). Wohlmöglich wurden immer wieder ähnliche Systeme entwickelt und lokal eingesetzt, wohingegen sich noch kaum ein System etabliert (Strickroth & Pinkwart, 2017). Strickroth und Pinkwart (2017) haben besondere Ansätze untersucht und Trends identifiziert.

3.2.1 Klassifizierung nach Generation

Ausgehend von einer ersten Publikation in den 1960er Jahren bis 2005 haben Douce, Livingstone und Orwell (2005) Systeme zur automatischen textbasierten Bewertung untersucht und entsprechend des Alters in drei Generationen klassifiziert:

3.2.1.1 Erste Generation – frühe Bewertungssysteme

Beim frühsten Beispiel automatischer Bewertung reichten Studenten Programme in Assemblersprache auf gestanzten Karten ein, anstatt Compiler und Texteditoren zu verwenden (Douce et. al., 2005). Das Bewertungssystem las diese Karte ein und befand den eingestanzten Code entweder als falsch oder abgeschlossen (Douce et. al., 2005). Mit der Entwicklung der Programmiersysteme entwickelten sich auch die Bewertungssysteme. Bereits 1969 haben Hext und Winnings ein System eingeführt, mit welchem Programme durch einen Abgleich zwischen Testdaten und dem Output überprüft wurden (Douce et. al., 2005). Tools der ersten Generation zeigten schon damals Vorteile der besseren Nutzung der Ressourcen eines Tutors sowie die Möglichkeit die Anzahl der Studenten dank Computereffizienz zu erhöhen (Douce et. al., 2005).

3.2.1.2 Zweite Generation – Toolorientierte Systeme

Die entweder kommandozeilenbasierten oder GUI-basierten Tools der zweiten Generation vereinfachten die automatische Bewertung weiter, indem sie den Code nach differenzierteren Kriterien überprüfen und Reports generieren (Douce et. al., 2005). Das Funktionsspektrum fortgeschrittener Tools dieser Generation ging über das einfache Bewerten

hinaus und neben den Dozenten konnten auch Lernende die Systeme verwenden, um Vorlesungsnotizen sowie Aufgaben abzurufen (Douce et. al., 2005). Einige Systeme der zweiten Generation haben sich zu weborientierten Systemen der dritten Generation weiterentwickelt.

3.2.1.3 Dritte Generation – Weborientierte Systeme

Bewertungssysteme der dritten Generation nutzen die Entwicklungen der Web-Technologie und verfolgen gleichzeitig anspruchsvollere Testansätze (Douce et. al., 2005). Neben der Überprüfung der syntaktischen Korrektheit und der Funktionstüchtigkeit eines Programmes ermöglichen Systeme der dritten Generation weitere Testansätze wie die Überprüfung des Programm-Design (Douce et. al., 2005). Zur Design-Überprüfung gehören Kriterien bezüglich des Formats (Typographie, Struktur oder Vorhandensein bestimmter Merkmale) sowie Programmkomplexität und Ausführungszeit. Nicht nur die Möglichkeiten bei der Überprüfung wurden besser, auch die Qualität und Präsentation des Feedbacks nahm zu (Douce et. al., 2005). Gleiches gilt für die Verwaltung von Aufgaben und Lösungen (Douce et. al., 2005).

3.2.1.4 Vierte Generation – Chatbotorientierte Systeme?

Da sich einen Trendwechsel von webbasierten zu dialogbasierten Interfaces (vgl. dazu 3.1.1 Chatbot) beobachten lässt, liegt die Überlegung nahe, dass die Programmierlehre künftig ebenfalls von der Chatbot-Technologie beeinflusst wird. Nichtsdestotrotz mutmasst Jensen (2017) in seinem Ausblick zu automatischen Bewertungssystemen, dass die Chatbot-Technologie zu diesem Zweck noch nicht geeignet sei: «Kritisch sehe ich Versuche, dem System natürlichsprachlich wirkende Feedbacks zu entlocken (die zudem didaktisch Sinn ergeben müssen). Wir sind mindestens ein Jahrzehnt von einer solchen Entwicklung entfernt».

Auch mittels Recherche konnte keine vergleichbare Chatbot-Anwendung in der Programmierlehre gefunden werden. Da ein Chatbot aber nur das Interface zu einem Service und nicht der Service selber ist, müssten die Systeme dritter Generation nicht grundlegend verändert werden und nur die Möglichkeiten zur Interkation mit dem System angepasst werden. Hinzu kommen aber neue didaktische und natürlichsprachige Herausforderungen bezüglich der Formulierung von Hilfestellungen, damit diese menschenähnlich wirken (Jensen, 2017).

3.2.2 Features automatischer Bewertungstools

Ziel des Literaturreviews von Ihantola, Ahoniemi, Karavirta und Seppälä (2010) ist es, die Features automatischer Bewertungstools für Programmierübungen darzustellen. Nachfolgend werden die für diese Arbeit relevanten Features aufgeführt.

Unterstützte Programmiersprachen: Ein Grossteil der Systeme ist für Java konzipiert, jedoch wird in einigen Systemen auch C/C++, Python, Pascal sowie Assembler und Shell Scripts unterstützt (Ihantola et al., 2010). Darüber hinaus gibt es auch Systeme, die unabhängig der Sprache zum Beispiel für Blackbox-Tests verwendet werden, wobei ein Input vorgegeben wird und der entsprechende Output untersucht wird (Ihantola et al., 2010).

Definition der Tests: Bewertungstools helfen bei der Erstellung und Definition von Tests (Ihantola et al., 2010). Tests können entweder aus der Industrie bezogen werden oder als spezialisierte Lösungen erstellt werden (Ihantola et al., 2010).

Erneute Einreichung von Lösungen: Mehrfacheinreichungen werden von den Systemen unterschiedlich gehandhabt, wobei viele Systeme die Anzahl an Einreichung limitieren (Ihantola et al., 2010). Als weitere Ansätze zählen Ihantola et al. (2010) die Einschränkung des Feedbacks oder Zeitstrafen für fehlgeschlagene Tests.

Möglichkeiten der manuellen Bewertung: Die Möglichkeiten zur manuellen Bewertung lassen sich in keine Eingriffsmöglichkeit, rein manuelle Bewertung sowie eine Kombination daraus kategorisieren (Ihantola et al., 2010).

Sandbox: Da der Code zur dynamischen Überprüfung ausgeführt werden muss, sollte die Code-Analyse in einer geschützten Umgebung stattfinden und zuvor mittels statischer Analyse nach schädlichem Code überprüft werden (Ihantola et al., 2010). Bei einer solchen geschützten Umgebung spricht man von einer Sandbox. Eine Sandbox kann auch auf einen separaten Server gesondert werden (Ihantola et al., 2010).

Verbreitung und Verfügbarkeit: Viele der untersuchten Systeme sind entsprechend Ihantola et al. (2010) nicht unter Open-Source-Lizenz veröffentlicht worden und können daher nicht im Internet gefunden werden (Ihantola et al., 2010).

3.2.3 Adaptive Feedbackansätze

Adaptives Feedback ist gemäss Le (2016) dynamisch und ermöglicht veränderbare, beziehungsweise angepasste Rückmeldungen. Die Adaptivität bezieht sich hierbei auf unterschiedliche Aspekte, wie beispielsweise die individuellen Eigenschaften des Lerners, die Aufgabe oder die Umgebung. Le (2016) kategorisiert adaptive Feedbackansätze in folgende Gruppen:

Ja / Nein-Feedback: Diese Form des Feedbacks gilt als grundlegendste, da sie lediglich angibt ob eine Aufgabe korrekt gelöst wurde oder nicht. Le (2016) zählt hierzu auch Antworten wie «Es liegen Syntaxfehler vor» oder «Es gibt keine Syntaxfehler, aber die Berechnung ist falsch».

Syntaxfeedback: Das Syntaxfeedback basiert auf der Ausgabe des Compilers und wird zumeist unverändert an den Lernenden weitergegeben, es gibt jedoch auch Ansätze in welchen eine umfangreichere Fehlerbeschreibung generiert wird (Le, 2016).

Semantisches Feedback: Beim semantischen Feedback bezieht sich ein Hinweis auf Fehler in der Erfüllung von Anforderungen (Le, 2016). Für diese Arbeit ist insbesondere der datengetriebene Ansatz interessant. Hierbei wird die Lösung eines Programms in Form des Outputs mit existierenden Lösungen aus einem Lösungsraum abgeglichen und bei Unterschieden auf mögliche Fehlerquellen hingewiesen (Le, 2016).

Qualitätsfeedback: Ein Qualitätsfeedback misst nicht nur ob ein Algorithmus korrekt ist, sondern auch dessen Effizienz hinsichtlich der Zeit und des verwendeten Speichers (Le, 2016). Ebenfalls gehören schlechte Programmierpraktiken wie „if (a==true)“ anstatt von „if (a)“ zu dieser Kategorie (Le, 2016).

3.2.4 Summative Feedbackansätze

Anschliessend werden summative Feedbacktypen nach Keuning, Jeuring und Heeren (2016) aufgeführt und untersucht. Summatives Feedback bezieht sich auf die Beurteilung des Erfüllungsgrad nach Abgabe einer Lösung (Keuning et al., 2016).

- Kenntnis des Erfolgs für eine Menge von Aufgaben (z. B. 85% korrekt)
- Kenntnis des Ergebnisses bzw. der Antworten (z. B. erfüllt alle Testfälle / Constraints, Ausgabe entspricht der Musterlösung)
- Kenntnis eines korrekten Ergebnisses (z. B. Darstellung einer Musterlösung)

- Kenntnis über Aufgabenbeschränkungen (z. B. Hinweise zu Anforderungen: for-Schleife muss genutzt werden)
- Kenntnis über Konzepte (z. B. weitergehende Erklärungen oder Beispiele)
- Kenntnis über Fehler (z. B. Testfehlschläge, Syntaxfehler, Laufzeit- bzw. logische Fehler, Styleverletzungen, Performanzprobleme).
- Kenntnis darüber, wie fortzufahren ist (z. B. Hinweise, wie ein Fehler behoben werden kann oder wie die aktuelle Lösung ergänzt werden sollte).
- Kenntnis über Metakognition (z. B. Überprüfung, ob ein Lernender weiß, warum eine Lösung korrekt ist)

Bei ihrer Untersuchung haben Keuning et al. (2016) festgestellt, dass 60% der untersuchten Systeme jeweils nur einen dieser Feedbacktypen unterstützen. Als Erklärung dafür nennen sie die Konzipierung als reine Bewertungssysteme und die Auslegung auf grosse Nutzerzahlen. Daher scheint deren Fokus nicht auf detailliertem Feedback zu liegen (Keuning et al., 2016). Nichtsdestotrotz geben viele Autoren an, dass mit den Systemen der Lernerfolg verbessert werden soll, wofür aber nach Ansicht von Keuning et al. (2016) eine einfache Auflistung von Fehlern nicht ausreicht.

3.2.5 Implikationen aus einer akademischen Anwendung

Striewe und Garmann (2017) stellen auf Basis des Einsatzes von automatischen Systemen zur Programmbewertung an der Universität Duisburg-Essen sowie der Hochschule Hannover verschiedene Implikationen vor. Das Beispielszenario basiert auf dem Einsatz der Systeme JACK an der Universität Duisburg-Essen sowie GRAJA an der Hochschule Hannover, welche jeweils in der Einführungsvorlesung zur objektorientierten Programmierung in Java zum Einsatz kommen (Striewe & Garmann, 2017).

Die automatische Bewertung erfordert grundlegende Eigenschaften der Lösung wie Klassen- und Methodenbezeichnungen (Striewe & Garmann, 2017). Weil sich der Lernende dadurch bei der Lösung an einige Vorgaben halten muss und ihm einige Entwurfsentscheidungen vorweggenommen werden, verringern sich die Freiheitsgrade bei der Programmierung (Striewe & Garmann, 2017). Striewe und Garmann (2017) sehen daher die Grenzen der automatischen Bewertung dort, «wo die kreative Leistung eines Studierenden wichtiger ist als die Umsetzung formaler Regeln». In der Programmierung gibt es beliebig viele korrekte Lösungen und demzufolge auch beliebig viele Möglichkeiten einen Fehler zu beheben (Striewe & Garmann, 2017).

Striewe und Garmann (2017) haben sowohl Erfahrungswerte von Dozenten und Studenten gesammelt. Dozenten werden zwar durch die Tools entlasten, gleichwohl kann die menschliche Betreuung nicht gänzlich abgeschafft werden. Die verfügbare Zeit der Dozenten kann dafür für wichtigere Aspekte wie die Betreuung von schwächeren Studenten eingesetzt werden. Obschon ein Tutor ein differenzierteres Feedback geben kann, sehen Striewe und Garmann (2017) den Vorteil der automatischen Bewertung in der Objektivität und der kürzeren Latenz zwischen Abgabe und Bewertung. Studenten hingegen empfanden das Bewertungssystem als hilfreich und im Gegensatz zur menschlichen Bewertung wird die maschinelle «als schneller, vergleichbar gut beim Entdecken von Fehlern, jedoch nicht als gerechter eingeschätzt» (Striewe & Garmann, 2017). Der wohl wichtigste Aspekt der Untersuchung von Striewe und Garmann (2017) ist derjenige, dass dank der Einführung automatischer Bewertungssysteme der Anteil an Studenten, welche die Lehrveranstaltung erfolgreich abschlossen, erheblich gesteigert werden konnte.

3.3 Fachlicher Kontext

Als fachlicher Kontext dient der Kurs Software Engineering an der SML. Darin wird ein E-Learning-Angebot bestehend aus Moodle als Kursmanagementplattform und repl.it als Programmierumgebung genutzt. Auf Grund dessen, dass das Artefakt dieser Arbeit darauf aufbaut, werden Moodle und repl.it nachfolgend vorgestellt.

3.3.1 Moodle Seite

Die Open-Source-Plattform Moodle wird als Kursmanagementsystem und Lernplattform verwendet. Kursmaterialen als Links, Texte und Dokumente werden darüber ausgetauscht und Lernaktivitäten in Form von Foren und Aufgaben bereitgestellt. Das weitverbreitete Lernmanagementsystem bietet bereits von sich aus Aufgabenformate wie Multiple Choice und Fill-in, jedoch keine Features zur automatischen Bewertung von Programmieraufgaben (Fricke & Striewe, 2017, S. 280).

3.3.2 Repl.it Server

Weil Moodle von sich aus keine Programmieraufgaben unterstützt, wird das IDE repl.it als Programmierumgebung verwendet. Während traditionelle IDEs meistens einen Software-Download bedingen, erlaubt es repl.it ohne die Notwendigkeit einer komplizierten Installation oder Setup-Prozedur Code direkt im Browser zu schreiben und auszuführen (Lardinois, 2018). Dabei unterstützt repl.it die meisten Programmiersprachen sowie

die gängigsten Frameworks (Lardinois, 2018). Mit den gleichnamigen Repls bietet repl.it Arbeitsbereiche als Container auf einer virtuellen Maschine an, in welchen Code dank dem integrierten Compiler «sandboxed» ausgeführt werden kann (repl.it, o. J.).

Dank der Plattformunabhängigkeit kann repl.it mit jedem internetfähigen Device verwendet werden, was repl.it zu einem beliebten Tool in der Programmierlehre macht (repl.it, o. J.). Ihren schulischen und akademischen Mehrwert hat Repl.it verstanden und dahingehend ihr Produktportfolio mit dem kostenpflichtigen Classroom erweitert (repl.it, o. J.). Ein Classroom ist eine Kollektion von Aufgaben, welche ihrerseits Assignments genannt werden (repl.it, o. J.). Wie es der Name vermuten lässt, befinden sich in einem «Klassenzimmer» sowohl Lehrer (Teacher), die für die Erstellung und Verwaltung von Aufgaben zuständig sind, als auch Studenten (Students), welche diese Aufgaben lösen.

Repl.it bietet mehrere Möglichkeiten zur manuellen oder automatischen Bewertung studentischer Abgaben. Bei der manuellen Bewertung muss der Dozent jede Abgabe händisch überprüfen und beurteilen, ob sie korrekt gelöst wurde. Für die automatische Bewertung gibt es wiederum zwei Möglichkeiten. Mit dem I/O-Matching lässt sich der Output eines Programms automatisch mit zuvor festgelegten Werten abgleichen und so eine Aussage darüber gemacht werden, ob der Code valide ist (repl.it, o. J.). Als zweite Variante zur automatischen Code-Analyse erlaubt es repl.it Unit-Tests zu schreiben (repl.it, o. J.). Mit diesen Tests kann der Code auf Methodenebene nach funktionaler Korrektheit überprüft werden (repl.it, o. J.).

Wichtig ist hierbei anzumerken, dass die Bewertung durch repl.it unabhängig der Bewertung durch den Helper-Bot stattfindet. Obschon der Helper-Bot sehr wahrscheinlich als Reaktion auf eine negative Bewertung durch repl.it aufgerufen wird, findet gleichwohl eine separate Code-Überprüfung durch den Helper-Bot statt.

4 Anforderungen an den Helper-Bot

Die in diesem Kapitel beschriebenen Anforderungen ergeben sich sowohl aus den Forschungsfragen, wie auch aus der vorhergehenden Analyse der Ausgangslage. Dabei fliessen relevante Ansätze aus dem Forschungsstand unter Berücksichtigung des fachlichen Kontextes in die Anforderungsdefinition mit ein. Ziel und Zweck dieses Kapitels ist es die Forschungsfragen durch die Erkenntnisse aus der Ausgangslage anzureichern und für die Realisierung des Helper-Bots zu konkretisieren.

Die Anforderungsbeschreibung findet auf Basis von Abbildung 1 statt, welche die vereinfachte Systemstruktur des Helper-Bots zeigt. Darin sind einerseits die bereits bestehenden (blau) sowie die zu erstellenden Teilsysteme abgebildet. Ebenso zeigt Abbildung 1 mit den nummerierten Schritten 1) bis 3) die grundlegende Funktionalität des Helper-Bots. Die dahinterstehende User-Story wird zunächst erläutert. Verschiedene Teile dieser User-Story müssen weiter untersucht werden:

> ***Grundlende User-Story**: Ein Student will sich bei der Erarbeitung von Programmierübungen selbständig und ohne Zeitverzögerung Hilfe besorgen, indem er sein Anliegen in natürlicher Sprache formulieren kann.*

Erarbeitung von Programmierübungen: Soll weiterhin über Moodle stattfinden, wozu repl.it-Aufgaben als IFrame integriert werden soll. Damit der Helper-Bot zu einer Aufgabe Hilfe leisten kann, muss diese zuvor abgegeben werden (1), wobei ein Webhook ausgelöst und die Lösung an den Helper-Bot übermittelt wird.

Selbständig und ohne Zeitverzögerung Hilfe besorgen: Braucht der Student Hilfe bei der Lösung einer Aufgabe, will er umgehend Hilfe und nicht erst auf die Antwort eines Dozenten warten. Dazu hat er die Möglichkeit den Helper-Bot aufzurufen (2). Damit der Helper-Bot Hilfe leisten kann, muss er Zugang zum studentischen Programm haben, das Programm durch Ausführung überprüfen und sinnvolles Feedback generieren.

In natürlicher Sprache: Die Kommunikation erfolgt über menschliche Sprache, wobei der Helper-Bot versucht die Benutzereingaben zu verstehen und entsprechende Hilfeleitungen zu erzeugen.

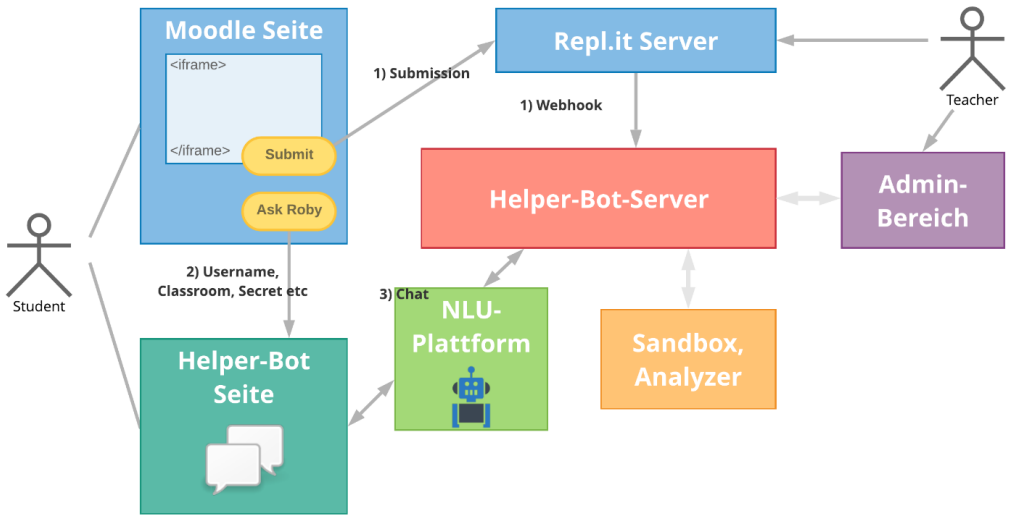


Abbildung 1: Vereinfachte Systemarchitektur als Ausgangslage

4.1 Zerlegung in Komponenten

Die Darstellung der Systemarchitektur in Abbildung 1 veranschaulicht das Zusammenspiel mehrerer Komponenten die als Stumme den Helper-Bot ausmachen. Um die Gesamtkomplexität zu reduzieren, wird der Helper-Bot zur Anforderungsdefinition in die einzelnen Komponenten zerlegt. Mit der Zerlegung des Systems in mehrere Teilsysteme werden die Anforderungen des Helper-Bot an die einzelnen Teilsysteme weitergegeben. Gleichwohl besitzen die Komponenten Interdependenzen und können nicht isoliert voneinander umgesetzt werden.

4.2 Anforderungsdefinition auf Komponentenebene

Hiernach werden die einzelnen Komponenten kurz eingeführt und deren Anforderungen erläutert. Ein umfassender Anforderungskatalog befindet sich in Tabelle 2 im Anhang A.

Moodle-Seite: Auf Moodle muss ein Test-Kurs und entsprechende Kursseiten angelegt werden, in welche eine Programmieraufgabe als iFrame eingebettet werden kann. Weiter muss es auf der jeweiligen Kursseite eine Möglichkeit geben den Helper-Bot aufzurufen und Meta-Daten wie Student und Aufgabenummer zu übermitteln.

Repl.it: Da die Erstellung und Verwaltung von Programmeraufgaben nicht Teil dieser Arbeit ist, muss auf repl.it lediglich ein Webhook eingerichtet werden, welcher die Submissions an den Helper-Bot sendet.

Helper-Bot-Server: Der Helper-Bot-Server gilt als Dreh- und Angelpunkt des Gesamtsystems. Seine Hauptaufgabe liegt darin, Anfragen der anderen Komponenten entgegenzunehmen, diese an die entsprechende Ressource weiterzugeben und die übermittelten Daten zu persistieren.

NLU-Plattform: Ziel dieser Arbeit ist es nicht eine eigene NLU-Komponente zu erstellen, sondern zur Beantwortung der zweiten Forschungsfrage ein passendes Framework zu evaluieren. Der Chatbot muss fähig sein Benutzereingaben zu interpretieren und einen externen Service zur Implementierung der eigenen Logik anzusteuern. Zudem soll die Konversation didaktisch sinnvoll gestaltet sein.

Chat-Interface: Weiter muss der Student über ein Chat-Interface mit dem Helper-Bot interagieren können. Hierzu gibt es Plattformen wie Facebook Messenger oder Telegram. Alternativ bieten gewisse NLU-Plattformen eigene Interfaces an oder es kann ein eigenes entworfen werden. Eine passende Lösung muss in die Anwendung eingebaut werden.

Code-Analyse: Damit der Chatbot Hilfestellung zu einer Aufgabe geben kann, muss der Code überprüft werden, wofür es wie in der Ausgangslage beschrieben, verschiedene Ansätze gibt. Grundsätzlich soll der Helper-Bot den Code sowohl statisch wie auch dynamisch überprüfen und bei Fehler entsprechendes Feedback generieren. Da der Code zur dynamischen Überprüfung ausgeführt werden muss, sollte die Überprüfung in einer gesicherten «Sandbox» stattfinden.

Admin-Bereich: Der Admin-Bereich dient zum Zwecke der Konfiguration und Parametrisierung des Helper-Bots sowie zum Abruf von Daten wie Studenten, Assignments, Konversationen usw. Welche Ansichten in der akademischen Praxis sinnvoll sind, gilt es hinsichtlich der fünften Forschungsfrage zu klären. Da diese Daten nur für die Dozenten bestimmt sind und aus Gründen des Datenschutzes, muss der Admin-Bereich passwortgeschützt sein.

Der Realisierung des Artefaktes obliegt der Umsetzung der einzelnen Komponenten mit ihren Anforderungen. Obschon mit der Anforderungsdefinition die grundlegende Funktionalität und Architektur des Helper-Bots vorgegeben ist, gibt es bezüglich der Systemarchitektur zu Beginn noch einige Unklarheiten. Die Entscheidungen und Überlegungen die zur finalen Systemarchitektur geführt haben, werden im Kapitel 6 behandelt.

5 Evaluation einer NLU-Plattform

In dieser Arbeit wird keine eigene Komponente zur Sprachverarbeitung entwickelt, sondern ein passendes Framework evaluiert und implementiert. Dazu gibt es verschiedene cloudbasierte Lösungen, die durch Parametrisierung und Training, so konfiguriert werden können, dass sie zu bestimmten Benutzereingaben entsprechend reagieren. Vier dieser Plattformen wurden zur Beantwortung der zweiten Forschungsfrage untersucht und bewertet. Obschon sie einige gemeinsame Funktionen bieten, unterschieden sie sich in gewissen Aspekten erheblich voneinander (Canonico & De Russis, 2018). Die Untersuchung besteht aus einer Einführung in die jeweilige Plattform sowie einem tabellarischen Vergleich in Tabelle 3 im Anhang B. Der Evaluation liegen die Untersuchungen von NLU-Plattformen von Canonico und De Russis (2018) und Langer (2018) zugrunde.

5.1 Plattformen

5.1.1 Dialogflow

Dialogflow als Nachfolger von API.AI ist die von Google betriebene NLU-Plattform zur Erstellung von Chatbots. Es wird zwischen einer Standard- und einer Business-Variante unterschieden, welche sich bezüglich des Umfangs des Supports und der Einbettung in die Google Cloud-Plattform unterscheiden (Langer, 2018).

Die Erstellung und Verwaltung eines Chatbots erfolgen über eine webbasierte Anwendung und über mehrere Formulare, welche jeweils einen Teil der Konversation darstellen. Ein solches Formular besteht aus einem Intent sowie einigen Beispielsätzen, die dazu verwendet werden, die NLU-Komponente zu trainieren (Canonico & De Russis, 2018). Die Sprachverarbeitung kann neben Intents auch Entitäten aus einer Benutzereingabe auslesen (Canonico & De Russis, 2018). Ebenso wird Conversation Management unterstützt (Canonico & De Russis, 2018).

Neben zahlreichen Integrationsmöglichkeiten gängiger Chat-Plattformen, besitzt Dialogflow auch Schnittstellen zur Erstellung eines eigenen Interfaces (Langer, 2018). Für die Verwendung mehrerer Plattformen bietet Dialogflow zudem die Möglichkeit, plattformspezifische Antworten zu definieren (Langer, 2018). Ebenfalls erlaubt es Dialogflow einen Service zur Implementierung der eigenen Logik anzusteuern (Langer, 2018).

5.1.2 Wit.ai

Wit.ai ist diejenige Plattform, die von Facebook betrieben wird. Wie Dialogflow kann auch wit.ai in der Standard-Version kostenlos verwendet werden. Da sich wit.ai vor Allem auf die Extraktion von Bedeutungen aus einem Satz konzentriert, fungiert diese Plattform viel eher als NLU-Parser als eine NLU-Plattform (Canonico & De Russis, 2018). Dies zeigt sich auch daran, dass wit.ai keine direkte Frontend-Anbindung mit Chat-Plattformen anbietet und keine Conversation Management unterstützt (Canonico & De Russis, 2018). All diese zusätzlichen Funktionen obliegen dem Entwickler und müssen selber integriert oder realisiert werden (Canonico & De Russis, 2018).

5.1.3 IBM Watson Assistent

Der IBM Watson Assistent ist Teil des IBM Cloud-Services und erlaubt die Erstellung von Chatbots (Canonico & De Russis, 2018). Basierend auf einem neuronalen Netzwerk, kann der Watson Assistent Absichten verstehen sowie Entitäten und Dialoge interpretieren (Canonico & De Russis, 2018). Für das Erreichen einer hohen Interpretationsgenauigkeit versucht der Watson Assistent möglichst viel Kontext in die Verarbeitung miteinzubeziehen, wofür ihm neben dem Inhalt der Eingabe eine Wissensbasis, bestehend aus einer Milliarde Wikipedia-Wörtern zur Verfügung steht (Canonico & De Russis, 2018).

Der Watson Assistent unterstützt 13 Sprachen (Langer, 2018). Da es nur wenige vordefinierte Entitäten gibt, müssen diese zum grössten Teil selber vom Ersteller trainiert werden (Langer, 2018). Ebenfalls besitzt der Watson Assistent keine direkte Anbindung an Chat-Plattformen, jedoch kann Twilio als Middleware dazu verwendet werden (Langer, 2018). Der Watson Assistent kann ein externes Backend-System auf Grundlage der Benutzerabsichten aufrufen und damit interagieren (Langer, 2018).

5.1.4 Amazon Lex

Die Sprachverarbeitung von Amazon Lex als Teil der Amazon Web Services unterstützt nur Englisch (Canonico & De Russis, 2018). Dafür bietet Amazon Lex fortschrittlichen Deep-Learning-Funktionen zur Umwandlung von Sprache in Text und zur Erkennung von Absichten (Canonico & De Russis, 2018). Dabei stehen Amazon Lex dieselben Technologien zur Verfügung, die auch den Kern von Amazon Alexa ausmachen (Canonico & De Russis, 2018). Amazon Lex besitzt daher ein grosses Spektrum an vorgefertigten Entitäten (Canonico & De Russis, 2018).

5.2 Bewertungskriterien

Der tabellarische Vergleich der NLU-Plattformen befindet sich im Anhang B. Hiernach werden die darin bewerteten Kriterien erläutert.

Usability: Das Kriterium Usability wurde von Canonico und De Russis (2018) übernommen und zeigt die wahrgenommene Benutzerfreundlichkeit, wobei zwischen Hoch (einfach und intuitiv für einen Entwickler), Mittel und Niedrig (schwierig zu bedienen und fehlende Dokumentation) unterschieden wird.

Spracherkennung: Befasst sich einerseits mit den unterstützten Sprachen sowie mit der Erkennung von Intents und Entities (vgl. dazu Kapitel 3.1.1.2). Ebenfalls unter diesen Punkt fällt der Aspekt des Conversation Managements (vgl. dazu Kapitel 3.1.1.5)

Frontend-Anbindung: Welche Front-Anbindungsmöglichkeiten bietet die Plattform. Die Wahl einer geeigneten Chat-Plattform bedarf nicht nur technische Aspekte, sondern gleichermassen Gestaltungsüberlegungen (z.B. Rich-Contents) und die Berücksichtigung der Benutzerfreundlichkeit (Shevat, 2017, S. 44ff.). Trotz der Vielzahl an verschiedenen Plattformen rät Shevat (2017, S. 44ff.) gleichwohl, sich anfangs auf eine Plattform zu konzentrieren.

Backend-Anbindung: Darunter fallen sämtliche Webhooks und SDKs, welche es erlauben, den Chatbot in eine separate Software zu integrieren. Eine solche Integration ist zentral, da über ein Backend die Logik des Helper-Bots in die Konversation implementiert werden muss. Daher ist dieses Kriterium ausschlaggebend für die Wahl einer Plattform.

Preis: Der Preis für die Nutzung der NLU-Plattform. Wird meistens auf Basis der Anzahl an Aufrufen berechnet. Viele Plattformen unterscheiden zwischen Standard- und Business-Version.

5.3 Evaluation einer geeigneten Plattform

Hinsichtlich der Wahl einer Plattform müssen die Anforderungen **A8**, **A9** und **A10** berücksichtigt werden. Eine Komponente zur Sprachverarbeitung besitzen alle Plattformen, jedoch unterstützen nicht alle Plattformen Deutsch als Sprache. Daher ist Amazon Lex nicht für den Helper-Bot geeignet. Von den drei verbliebenen Plattformen enthält Dialogflow

am meisten vordefinierte Entitäten und beinhaltet darüber hinaus noch Kontext-Management.

Bezüglich der Anbindung an einen externen Service bieten alle Plattformen zumindest SDKs und Dialogflow zusätzlich noch Webhooks. Grundsätzlich erlauben es SDKs, respektive die darin enthaltenen APIs und Webhooks Daten zwischen zwei Anwendungen auszutauschen. Inwiefern sich APIs von Webhooks unterscheiden ist im Kapitel 3.1.3 beschrieben. Webhooks dürften gegenüber den APIs Vorteile in der einfacheren Bedingung haben.

Dialogflow zeigt sowohl Vorteile bei der Sprachverarbeitung als auch bei der Integration eines externen Service. Auch die limitierte Aufrufzahl in der Standardversion dürfte vorerst kein Problem darstellen. Für die Realisierung des Helper-Bots wird daher die NLU-Plattform Dialogflow verwendet.

5.4 Konkretisierung der Anforderungen

Mit der Evaluation einer NLU-Plattform lässt sich die Anforderung der Anbindung eines externen Service (**A9**) konkretisieren. Wie im vorgehenden Kapitel beschrieben, ermöglicht es Dialogflow ein Backend über einen Webhook anzusteuern. Die dabei aufgerufene Ressource nennt Dialogflow Fulfillment. «Fulfillment is code that's deployed as a webhook that lets your Dialogflow agent call business logic on an intent-by-intent basis» (Dialogflow, o. J.). Damit ermöglicht es ein Fulfillment während einer Konversation, die aus den Benutzereingaben extrahierten Informationen zu nutzen, um dynamische Reaktionen zu erzeugen oder Aktionen auf dem Helper-Bot auszulösen (Dialogflow, o. J.).

Der Helper-Bot muss fähig sein eine Fulfillment-Anfrage von Dialogflow entgegenzunehmen, die entsprechende Ressource aufzurufen und dynamische Antworten zurück an Dialogflow zu senden. Für jeden unterstützen Intent, beispielsweise Hilfe anfordern, muss dazu eine Ressource erstellt werden. Inwiefern sich die Kommunikation zwischen Dialogflow und dem Helper-Bot gestaltet, wird im nächsten Kapitel behandelt.

6 Architekturdokumentation des Helper-Bots

Als Dokumentation zur Architektur, beschreibt dieses Kapitel diejenigen Entscheidungen, die sowohl auf Komponenten-Ebene, als auch auf Ebene des Gesamtsystems zentral für die Umsetzung des Helper-Bots waren. Es galt jeweils aus verschiedenen Lösungen die beste zu eruieren und zu realisieren. Dabei fand die Entscheidungsfindung dank dem inkrementellen Vorgehen jeweils auf Basis von Zwischenergebnissen statt. Auch dieses Kapitel orientiert sich an der inkrementellen Denkweise und beschreibt drei aufeinander aufbauende Stadien der Systemarchitektur. Zunächst werden im ersten Stadium die Grundstrukturen aufgezeigt, darauffolgend wird im zweiten Stadium die Implementierung der Codeanalyse beschrieben und zuletzt wird dokumentiert, wie der Chatbot mit dem Helper-Bot interagiert.

6.1 Grundstruktur und Admin-Bereich

Dem Helper-Bot liegt der Datenaustausch mit repl.it Dialogflow zugrunde. Einerseits werden Submissions von repl.it an den Helper-Bot gesendet und andererseits ist er für die Erfüllung von Fulfillments zuständig. In beiden Fällen findet der Austausch von Daten über Webhooks und in Form von JSON statt. Die grundlegende Funktionalität des Helper-Bots besteht nun darin, diese Daten entgegenzunehmen und die relevanten Informationen zu persistieren, so dass diese unter anderem im Admin-Bereich verfügbar sind. Diese Grundfunktionalitäten werden in diesem ersten Architekturabschnitt erörtert.

6.1.1 Laufzeitumgebung / Programmiersprache

Eine der ersten Entscheidungen, welche im Rahmen der Entwicklung einer Anwendung getätigt werden muss, ist die Wahl einer Programmiersprache. Hierbei spielen insbesondere Überlegungen hinsichtlich der Architektur und Anforderungen von Umsysteme eine Rolle. Gleichwohl geben weder die vorgegebenen Systeme Moodle und repl.it noch die gewählte NLU-Plattform Dialogflow die Verwendung einer Programmiersprache vor, so dass deren Wahl auf Basis anderer Kriterien stattfindet. Zwei wichtige Kriterien sind dabei, dass der Helper-Bot einerseits als Webanwendung geschrieben werden soll und dass andererseits der Prototyp zeitnah umgesetzt werden muss. Daher wurde die Laufzeitumgebung Node.js anderen serverseitigen Programmiersprachen wie Java oder PHP vorgezogen.

6.1.1.1 Node.js

Node.js oder kurz Node ist eine Laufzeitumgebung, in welcher JavaScript sowohl im Backend, als auch im Frontend verwendet wird (Tilkov & Vinoski, 2010). «Node.js's architecture makes it easy to use a highly expressive, functional language for server programming, without sacrificing performance and stepping out of the programming mainstream» (Tilkov & Vinoski, 2010). Aufgrund der zunehmenden Bekanntheit von Node.js, hat sich in den letzten Jahren eine beträchtliche Community sowie eine ganzes Ökosystem an Libraries für Node.js gebildet (Tilkov & Vinoski, 2010).

6.1.1.2 Express

Eine solche Library ist Express, welche auf den Webserver-Funktionalitäten von Node.js aufbaut, um APIs zu vereinfachen und nützliche Funktionen hinzuzufügen (Hahn, 2016, S. 27ff.). Um den Nutzen von Express aufzuzeigen, muss zunächst erklärt werden, inwiefern sich die Verarbeitung einer Benutzeranfrage durch Express von der reinen Node.js-Umgebung unterscheidet. Abbildung 2 zeigt dazu einen clientseitigen Aufruf und wie dieser von Node.js verarbeitet wird. Darin stellt der Kreis vom Programmierer entwickelter Code dar, wohingegen Quadrate nicht beeinflussbar sind (Hahn, 2016, S. 27ff.).

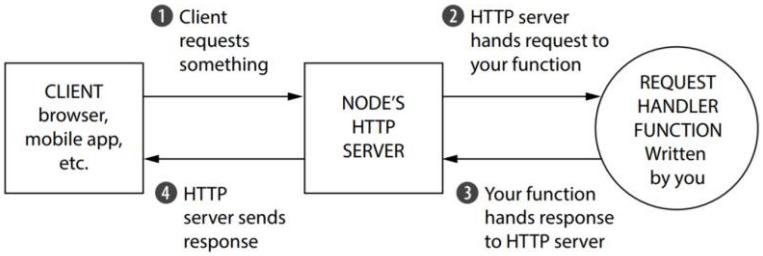


Abbildung 2: Ablauf eines Request in Node.js (Hahn, 2016)

Die Verarbeitung einer Anfrage in Node erfolgt über eine einzige Funktion, die als Request-Handler bezeichnet wird (Hahn, 2016, S. 27ff.). Der HTTP-Server von Node.js übernimmt hierbei die Verbindung zwischen Client und dem Request-Handler, sodass sich der Programmierer nicht um Netzwerkprotokolle kümmern muss (Hahn, 2016, S. 27ff.). Das Problem an der Node-Architektur ist jedoch, dass die Verarbeitung einer Anfrage durch eine einzelne Funktion schnell zu kompliziert wird und dass die Wiederverwendung von Funktion nicht möglich ist (Hahn, 2016, S. 27ff.).

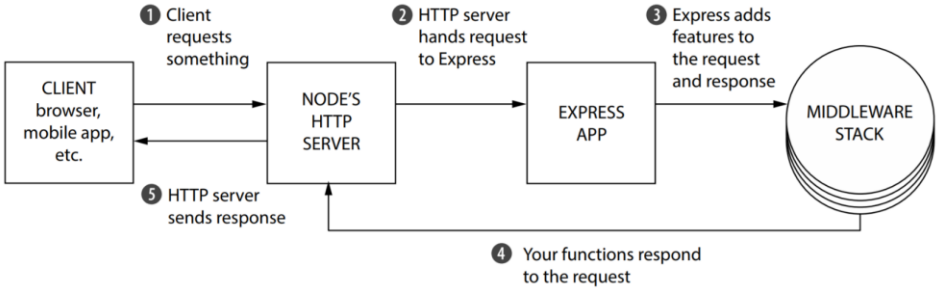


Abbildung 3: Ablauf eines Request mit Express (Hahn, 2016)

Obschon die in Abbildung 3 gezeigte Architektur von Express komplizierter wirkt, zeigen sich erhebliche Vorteile gegenüber der reinen Node-Umgebung. Anstelle eines einzelnen Request-Handlers erfolgt die Verarbeitung in Express über mehrere spezifische Funktionen (beispielsweise auch Third-Party-Funktionen) (Hahn, 2016, S. 27ff.). «Instead of managing one monolithic request handler function with verbose Node.js APIs, you write multiple small request handler functions that are made more pleasant by Express and its easier APIs» (Hahn, 2016, S. 27ff.).

Diese zwischengeschalteten Funktionen werden als **Middleware** bezeichnet. Beliebig viele solcher Middleware-Funktionen können zwischen der eingehenden Anfrage und der ausgehenden Antwort stehen, wobei jede dieser Funktonen die Response mit zusätzlichen Informationen anreichert (Hahn, 2016, S. 27ff.). Eine Middleware-Funktion eignet sich daher beispielsweise zur Überprüfung, ob ein Benutzer eingeloggt ist. Neben der Verwaltung der Middleware-Funktionen hilft Express dabei, die entsprechende Ressource zu einer Anfrage aufzurufen, was als **Routing** bezeichnet wird (Hahn, 2016, S. 27ff.). Eine solche Ressource ist jeweils abhängig von der URL und erfüllt einen bestimmten Zweck, wie beispielweise das Anzeigen des Login-Fensters.

Neben Express kommen weitere Node.js Libraries zum Einsatz. Die wichtigsten Libraries werden im Verlauf dieses Kapitels erwähnt und darüber hinaus befindet sich im Anhang C in Tabelle 4 eine Übersicht aller verwendeten Libraries.

6.1.2 Datenbank

Zum Persistieren der von repl.it und Dialogflow einhergehenden Daten braucht es eine Datenbank. Bezüglich der Wahl einer solchen stellt sich die Frage, ob diese relational oder nicht-relational (nicht nur relational) sein soll. Obwohl zumeist beide Optionen umsetzbar sind, gibt es einige wesentlichen Unterschiede, die es zu beachten gilt.

6.1.2.1 SQL vs. NoSQL

Bei relationalen **SQL**-Datenbanken wird Structured Query Language (SQL) verwendet, um Daten zu definieren und zu manipulieren (Ashwini, 2017). SQL als strukturierte Abfragesprache erfordert ein vordefiniertes Datenbankschema und sämtliche Daten müssen sich an dieses halten (Ashwini, 2017). Eine Änderung des Schemas stellt sich hierbei als kompliziert heraus und bedeutet zumeist schwerwiegende Änderungen für das gesamte System (Ashwini, 2017). Hierbei zeigen sich Vorteile von **NoSQL**-Datenbanken, welche sich nicht an ein starres Schema halten und stattdessen dynamische Schemata (spaltenorientiert, dokumentenorientiert, grafikbasiert usw.) für unstrukturierte Daten erlauben (Ashwini, 2017). Dafür fehlen NoSQL-Datenbanken Standards wie die ACID-Konformität (Atomicity, Consistency, Isolation, Durability) (Ashwini, 2017). Da die Datenstruktur grösstenteils durch repl.it vorgegeben ist und sich diese künftig nicht grundlegend ändern dürfte, wurde eine relationale einer nicht-relationalen Datenbank vorgezogen. Für den Helper-Bot wird daher eine MySQL-Datenbank verwendet.

6.1.2.2 Object-Relational mapping (ORM)

Um Daten aus einer relationalen Datenbank schnell und effektiv in einer objektorientierten Anwendung zu verwenden, gibt es sogenanntes Object Relation Mapping (ORM). ORM agiert hierbei als Interface zwischen zwei Systemen und hilft dabei Objekte einer Anwendung in eine relationale Datenbank abzulegen und auszulesen (Sequelize, o. J.). In dieser Arbeit wird das ORM Sequelize verwendet, welches diverse Feature wie Validierung, Synchronisierung und Assoziationen mit sich bringt und gleichzeitig die gängigsten Datenbanktypen wie PostgreSQL, MySQL und MariaDB unterstützt (Sequelize, o. J.).

6.1.3 Repl.it Webhook

Bei jeder Submission auf repl.it wird ein Webhook getriggert, Daten an den Endpoint versendet, die Anfrage von Node verarbeitet und via Express an die entsprechende Ressource weitergeleitet. Dort werden die Daten ausgelesen und in die Datenbank geschrieben. Zur Speicherung der Daten wird das Datenschema von repl.it weitgehendübernommen. Das entsprechende Datenbankschema ist in Abbildung 4 gezeigt. Zu jeder Submission gehört ein Student, ein Assignment, welches wiederum einem Classroom angehört sowie mehrere Files. Weitere Informationen zur repl.it-Schnittstelle können aus Tabelle 6 im Anhangs D entnommen werden.

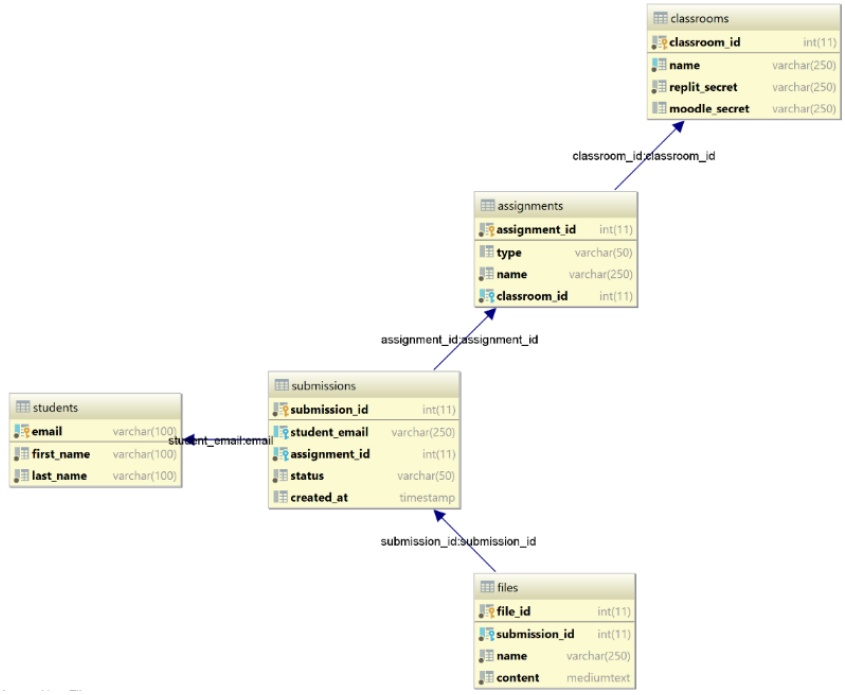


Abbildung 4: Grundlegendes Datenbankschema

6.1.4 Dialogflow Fulfillments

Da auch die Fulfillments (vgl. Kapitel 5.4) von Dialogflow auf einem Webhook basieren, erfolgt deren Verarbeitung nach vergleichbarem Prinzip wie bei den Submissions. Der Webhook wird immer dann ausgelöst, wenn ein Intent erkannt wird, welcher die Verarbeitung durch den Helper-Bots verlangt. Im Anhang D befindet sich dazu eine detaillierte Beschreibung der Schnittstell zwischen Dialogflow und dem Helper-Bot.

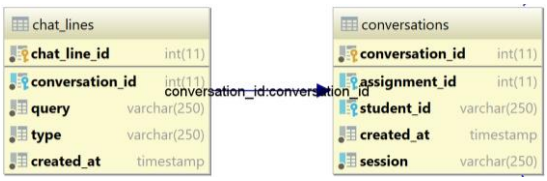


Abbildung 5: Datenbankschema zur Speicherung von Konversationen

Unabhängig des Erkannten Intents soll zunächst die Konversation gespeichert werden. Hierzu wird das Datenbankschema wie in Abbildung 5 gezeigt um zwei Tabellen erweitert. Jede Konversation besteht aus mehreren sogenannten Chat-Lines, wobei es sich jeweils um eine Benutzereingabe, respektive um eine Antwort des Chatbots handelt. Unterschieden werden diese beiden Arten von Nachrichten über das Attribut `type` (`request` oder `response`).

6.1.5 Admin-Bereich

Ebenfalls als Teil der Grundstruktur werden die Grundzüge des Admin-Bereichs implementiert. Über einen passwortgeschützten Bereich sollen die gespeicherten Daten abgerufen und die Schnittstellen zu den Umsystemen konfiguriert werden.

6.1.5.1 Passwortschutz

Als Ausgangslage für die Umsetzung des Passwortschutzes dient die Beschreibung von Debrah (2018), welcher unter Anwendung der Libraries Sequelize, PassportJS, und Express-Session eine Authentifizierung für Express realisiert. PassportJS knüpft am Konzept der Middleware von Express an und erlaubt es für bestimme REST-Anfragen Authentifizierungsschutz festzulegen. Sodann wird bei jedem Aufruf überprüft, ob in der aktuellen Session ein Benutzer eingeloggt ist. Für die Authentifizierung wird das Datenbankschema um die in Abbildung 6 gezeigt Tabelle `Users` erweitert.

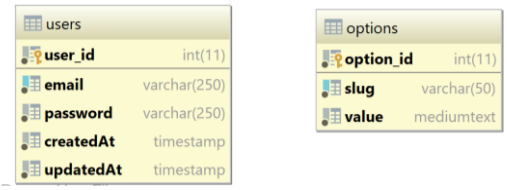


Abbildung 6: Datenbankschema zur Speicherung von Optionen und Users

6.1.5.2 Konfiguration und Parametrisierung

Für die Konfiguration und Parametrisierung müssen entsprechende Endpoints für GET- und POST-Anfragen definiert und Templates erstellt werden. Weiter muss für die Speicherung von Einstellungen die Datenbank um die in Abbildung 6 dargestellte Tabelle `options` erweitert werden. Diese beinhaltet Slug-Value-Paare, welche jeweils eine Konfigurationseinstellung darstellen.

6.1.6 Zusammengefasst

Bislang ist der Helper-Bot fähig Webhooks von repl.it und Dialogflow entgegenzunehmen, die übermittelten Daten in einer Datenbank zu schreiben und diese über den Admin-Bereich darzustellen. Ebenfalls findet die Konfiguration der Schnittstellen über den Admin-Bereich statt. Die entsprechende Architektur nach diesem ersten Abschnitt ist in Abbildung 7 grafisch zusammengefasst. Sie zeigt die vier Teilsysteme und wie diese mitei-

nander interagieren. Die vom Admin-Bereich ausgehenden gestrichelten Linien verweisen jeweils zu denjenigen Schnittstellen, welche über den Admin-Bereich konfiguriert werden. Da bislang noch kein Chat-Interface erstellt wurde, können Fulfillments nur über das Dialogflow-GUI ausgelöst werden, weshalb die entsprechende Linie durchbrochen dargestellt ist. Die für den Helper-Bot verwendete Verzeichnisstruktur sowie eine Beschreibung der einzelnen Verzeichnisse kann aus Tabelle 5 im Anhang C entnommen werden.

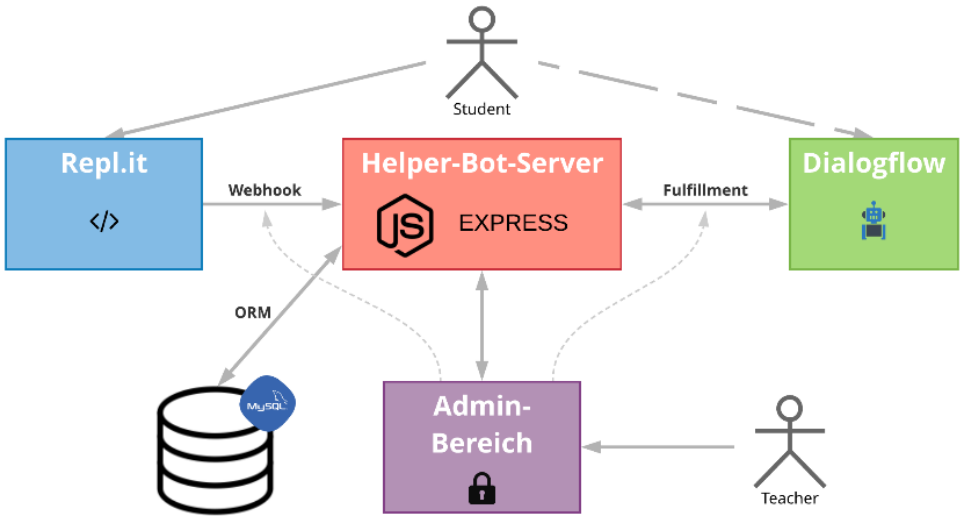


Abbildung 7: Grundlegende Anwendungsarchitektur

6.2 Code-Analyse als separate API

Für das statische und dynamische Code-Assessment muss ein Java-Programm kompiliert und interpretiert werden. Die Kompilierung findet dabei auf Ebene des Bytecodes statt, wohingegen der Code zur Interpretation ausgeführt werden muss. Zweiteres gestaltete sich insofern problematisch, da es zur Ausführung von Java eine Java-Laufzeitumgebung (Java Runtime Environment, kurz JRE) braucht, der Helper-Bot jedoch bislang auf Node basiert. Damit dennoch Java-Code ausgeführt und analysiert werden kann, gibt es mehrere Lösungen, die nachfolgend diskutiert werden.

Zunächst existieren mehrere Node.js Libraries, die den Support verschiedener Programmiersprachen in der Node-Umgebung versprechen. So auch die Libraries `Compile-Run` und `Compilex`, welche es entsprechend ihrer Beschreibungen erlauben, Java-Code zu kompilieren und auszuführen. Jedoch hat Compilex die aktive Entwicklung mit folgender Begründung eingestellt: «It started as an experiment and served its purpose well. But over

the time, new technologies like docker emerged. The existence of such technology and nature of node.js prove that compilex would not scale well and be production ready with its current state» (Bharathi, 2019). Die Verwendung einer solchen Library scheint demnach als ungeeignet, wohingegen aufkommende Technologien wie Docker und APIs bessere Ansätze versprechen (Bharathi, 2019). Während Docker (Anwendung zur Containervirtualisierung) für diese Arbeit als zu umfänglich erscheinen, zeigen sich APIs als valable Lösung für die Code-Analyse.

6.2.1 Eigenfertigung oder Fremdbezug einer API

Eine solche API kann sowohl als Dienst bezogen, als auch selber entwickelt werden. HackerEarth beispielsweise unterhält eine Code-API und stellt Endpunkte für die Kompilierung und Ausführung von Code in mehreren Sprachen zur Verfügung (HackerEarth, o. J.). Die API empfängt den Programm-Code per POST und gibt anschliessend Daten im JSON-Format zurück (HackerEarth, o. J.). Damit die HackerEarth-API verwendet werden kann, ist eine kostenpflichtige Registrierung erforderlich (HackerEarth, o. J.).

Für die Entwicklung einer eigenen Code-API ist die Inbetriebnahme eines zweiten Servers mit JRE notwendig. Wenngleich die Eigenentwicklung einen Mehraufwand bezüglich der Einrichtung und Konfiguration eines zweiten Servers bedeutet, bringt dieser Lösungsansatz Vorteile bezüglich der Kontrolle und Freiheit bei der Code-Überprüfung. So kann das statische und dynamische Assessment konkret auf die Bedürfnisse und Anforderungen des Helper-Bots angepasst werden. Bezüglich der Kontrolle gilt es insbesondere den Datenschutz zu beachten. Während studentischen Daten bei der Verwendung einer externen API an eine Drittpartei versendet werden, verlassen die Daten mit einer eigenen API nie den Kontrollraum. Aufgrund dieser beiden Aspekte scheint die Entwicklung einer eigenen Code-API für den Helper-Bot zweckmässiger zu sein.

6.2.2 Zentrale Feedbackgenerierung

Es wird immer dann eine Anfrage an die Code-API gesendet, wenn der Student Hilfe zu einer Aufgabe anfordert. Die Anfrage beinhaltet jeweils den zu überprüfenden Code und die durchzuführenden Tests. Mit der Code-API kann jedoch nur Java-Code ausgeführt und bewertet werden. Gleichwohl ist es denkbar, dass zukünftig weitere Programmiersprachen über den Helper-Bot angeboten werden. In diesem Fall müsste für jede Programmiersprache eine separate API geschrieben werden. Ihantola et al. (2010) haben mit

dem Feature `Unterstützte Programmiersprachen` auch einen Ansatz untersucht, bei welchen die Bewertung unabhängig der Programmiersprache stattfindet. Auf Basis eines Blackbox-Tests kann der Output nach zuvor definierten Inputs untersucht werden. Die Programmiersprache des Programms spielt dabei keine Rolle.

Zur Umsetzung dieses Ansatzes muss die Feedback-Generierung auf dem Helper-Bot-Server stattfinden und nicht auf der jeweiligen API. Das heisst die Code-API ist lediglich dafür zuständig den Code zu kompilieren und auszuführen und allfällige Fehler oder den Output zurückzugeben. Auf Seite des Helper-Bots wird, sofern kein Fehler aufgetreten ist, der Output untersucht und entsprechendes Feedback generiert. Die Schnittstelle zwischen dem Helper-Bot-Server und der Code-API ist im Anhang D in Tabelle 7 beschrieben.

6.2.3 Parametrisierung für die Feedbackgenerierung

Die jeweilige Hilfestellung zu einem Fehler obliegt der Parametrisierung durch einen Administrator. Diesbezüglich muss der Admin-Bereich sowohl Konfigurationsseiten für allgemeine Java-Fehler sowie aufgabenspezifische Logikfehler besitzen. Beide Fehlertypen werden über die gleichen Tabellen in Abbildung 8 gespeichert.

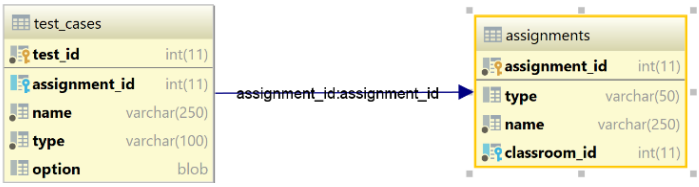


Abbildung 8: Datenbankschema für Testfälle

Die Attribute und deren Eigenschaften sind so gewählt, dass die Speicherung verschiedene Fehlertypen möglich ist. Unterschieden werden die Typen über das Attribut `type`. Weiter muss kein Fremdschlüssel zu den Assignments angegeben werden, sodass allgemeine und aufgabenspezifische Tests gleichermassen gespeichert werden können. Da der Inhalt des Attributes `option` jeweils Abhängig von der Fehlerart ist und die Werte sich grundlegend voneinander unterscheiden können, wurde dem Attribut der Typ `BLOB` zugewiesen. BLOB-Werte werden im Gegensatz zu anderen Feldtypen als binäre Zeichenketten behandelt und können daher beispielsweise Files enthalten. Dieser Feld-Typ wurde hinsichtlich der Erweiterung um Unit-Tests gewählt, sodass Java-Files darin gespeichert werden könnten.

6.2.4 Zusammengefasst

Abbildung 9 zeigt die um die Code-API ergänzte Systemarchitektur. Darin ist die Aufteilung zwischen der Code-Überprüfung und dem IO-Matching samt Feedback-Generierung ersichtlich. Das Programm wird an die API gesendet, dort kompiliert und ausgeführt und anschliessend zurück an den Helper-Bot gesendet, wo entsprechend des Outputs Feedback generiert wird. Bislang kann die Code-Überprüfung aber nur über das Backend und nicht direkt über einen Chatbot aufgerufen werden. Im nächsten und letzten Architekturabschnitt wird beschrieben, wie die Architektur in Abbildung 9 um die Chatbot-Technologie ergänzt wird.

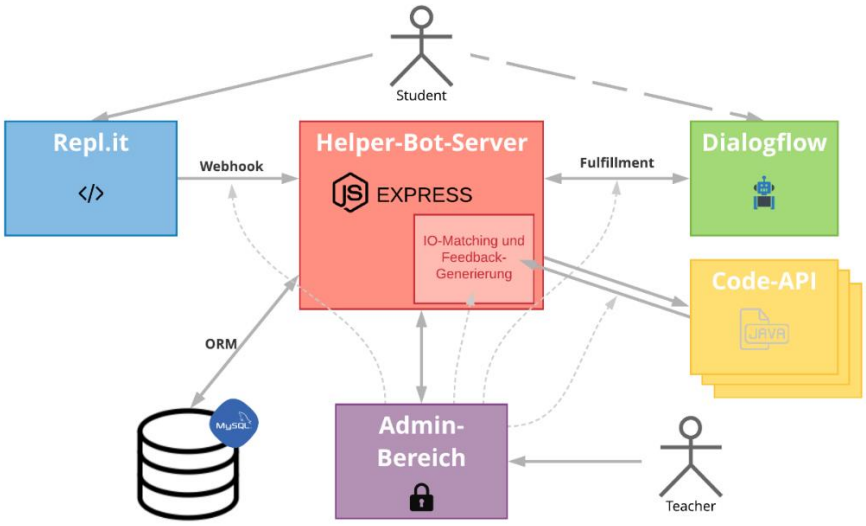


Abbildung 9: Systemarchitektur mit separater Code-API

6.3 Implementation des Chatbot

Nachfolgend wird die Architektur eines auf Dialogflow-Chatbots beschrieben und untersucht, inwiefern sich die Kommunikation zwischen Dialogflow und Helper-Bot umsetzen lässt. Diesbezüglich wurde bereits im ersten Abschnitt beschrieben, wie Fulfillments funktionieren und wie diese vom Helper-Bot behandelt werden.

6.3.1 Die vier Ebenen eines Dialogflow-Chatbots

Mohanoor (2019) unterteilt einen auf Dialogflow basierenden Chatbot in vier Ebenen. Die einzelnen Ebenen sind nicht strikt voneinander getrennt und deren Übergänge lassen sich teilweise nur schwer definieren (Mohanoor, 2019). Die vier Ebenen sind:

UI-Ebene: Das User-Interface, respektive das Chat-Interface ist diejenige Komponente eines Chatbots, mit welcher der Benutzer effektiv in Kontakt kommt. Die UI-Ebene ist dementsprechend die dem Benutzer am nächsten liegende Ebene (Mohanoor, 2019). Bezüglich des Interfaces sind daher grafische und konversationsbedingte Überlegungen wichtig.

Middleware/Integration Ebene: Die Middleware-, respektive Integrationsschicht verbindet die UI-Schicht mit dem NLU-Agenten (Mohanoor, 2019). Diese Ebene reagiert bei jeder Benutzereingabe, indem sie die Anfrage an die Funktion zur Erkennung der Absicht des NLU-Agenten sendet (Mohanoor, 2019). Dialogflow bietet hierzu One-Click Integration an und hilft dabei einen Chatbot auf einer oder mehreren Chat-Plattformen verfügbar zu machen (Dialogflow, o. J.). Ebenfalls kann ein Chatbot mit einem eigenen Interface interagieren.

Konversations-Ebene: Diese Ebene obliegt den Funktionalitäten von Dialogflow sowie der Parametrisierung von Absichten, Entitäten und Kontexten über die cloudbasierte Dialogflow-Plattform (Mohanoor, 2019). Diese Ebene kann also nur durch Parametrisierung beeinflusst werden.

Webhook/Fulfillment Ebene: Die eigentliche Logik einer Chatbot-Anwendung erfolgt über Fulfillments. Fulfillments wurden bereits im Kapitel 5.4 konkretisiert und in der ersten Architekturstufe verwendet.

Es sind nicht alle Ebenen zwingend notwendig (Mohanoor, 2019). Soll beispielsweise ein Chatbot als FAQ dienen, wird keine Fulfillment-Ebene benötigt, da statische Antworten genügen. Weiter kann die Umsetzung und Verwaltung der ersten drei Ebenen ausschliesslich über die Benutzeroberfläche von Dialogflow erfolgen und es muss prinzipiell keine Zeile Code geschrieben werden. Die grundlegende Architektur in Abbildung 7 basiert auf diesem Vorgehen und der Helper-Bot wird bislang nur zur Erfüllung von Fulfillments kontaktiert. Als Interface (samt Integration) wurde zunächst das Dialogflow-eigene Web-Interface verwendet.

6.3.2 Architekturprobleme

Während der Umsetzung der Anforderungen haben sich jedoch mehrere Probleme und Einschränkungen mit dieser grundlegenden Verwendung von Dialogflow offenbart. Ab-

bildung 10 zeigt dazu das Zusammenspiel der vier Chatbot-Ebenen sowie die Einschränkungen und deren Lösung. Zunächst werden die einzelnen Probleme erörtert und anschliessend eine Lösung präsentiert.

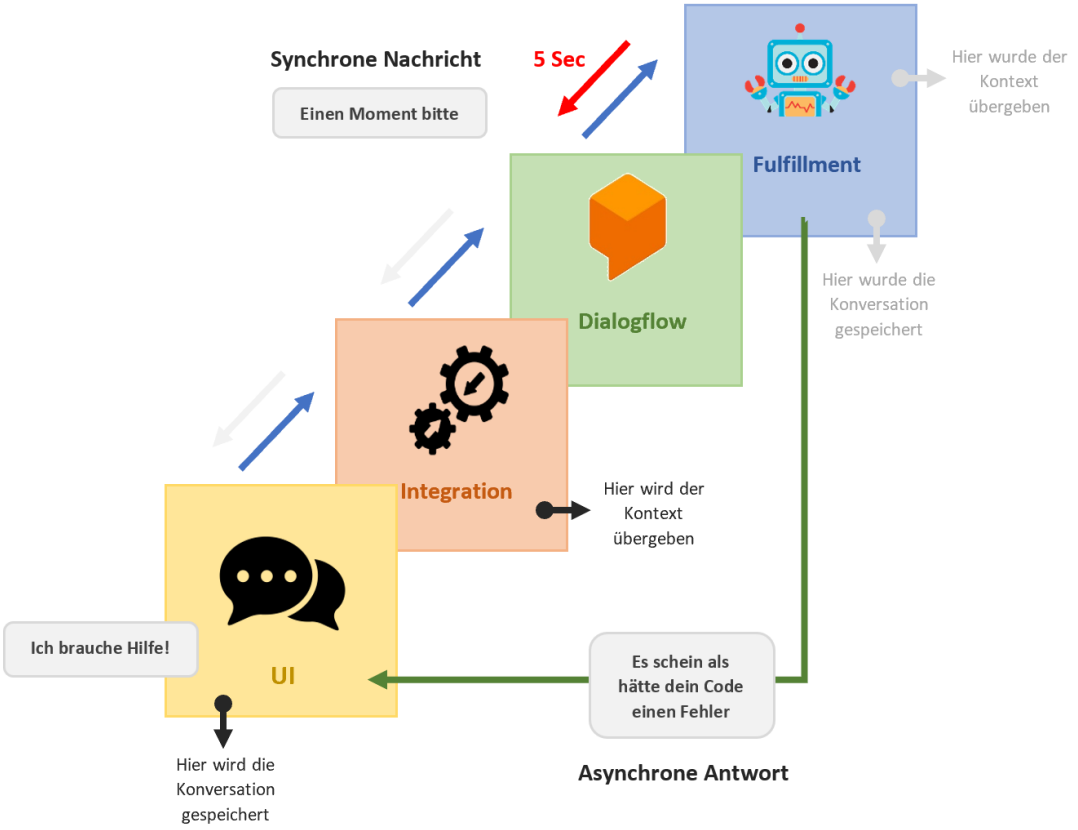


Abbildung 10: Die vier Ebenen eines Dialogflow-Chatbots

6.3.2.1 Slot-Filling und Fulfillments

Bislang werden Fulfillments, wie in Abbildung 10 beschrieben, dazu verwendet, Konversation zu speichern. Zu diesem Zweck muss für sämtliche Intents Fulfillments aktiviert sein. Nun verlangen es die Anforderungen, dass neben Intents auch Entities erfasst werden. Auf Konversationsebene können Entitäten einfach per Formular in der Dialogflow-Umgebung definiert werden. Anschliessend liest der Chatbot sämtliche Entitäten aus den Benutzereingaben aus und falls eine als notwendig markierte Entität fehlt, wird diese dank Kontextmanagement vom Chatbot erfragt. Dialogflow bezeichnet diesen Prozess als Slot-Filling. Nun zeigt sich aber ein Problem beim Persistieren von Konversationen, wenn Slot-Filling verwendet wird.

Dialogflow löst erst dann ein Fulfillment aus, wenn sämtliche Entitäten zu einem Intent bekannt sind. Beispielsweise auf die Absicht Hilfe zu erhalten, fragt der Chatbot zunächst, für welche Aufgabe Hilfe benötigt wird. Auf all diese Zwischenfragen hat der Helper-Bot aber keinen Zugriff und die Nachrichten können nicht gespeichert werden. Es sein denn, die Einstellung `Webhook für Slot-Filling` wird aktiviert. Dann wird nach dem Erkennen eines Intents für alle nachfolgenden Benutzereingaben ein Webhook ausgelöst, bis entweder alle Entitäten gesammelt oder ein neuer Intent erkannt wurde. So hat der Chatbot Zugang zu sämtlichen Zwischenfragen.

Jedoch führt Webhooks für Slot-Filling zu deutlich mehr Aufrufen und verbraucht unnötig Ressourcen. Ebenfalls hat sich während mehrerer Tests gezeigt, dass sich der Konversationsverlauf mit der Aktivierung von Webhooks für Slot-Filling kaum steuern lässt. Auch Mohanoor (2017) steht der Verwendung von Webhooks für Slot-Filling kritisch gegenüber und rät davon ab. Auch wenn es verschiedene Möglichkeiten zur Beeinflussung des Konversationsverlauf wie Kontext-Manipulation oder Follow-Up-Intents gibt, gestaltet sich deren Verarbeitung durch Dialogflow zumeist als intransparent und unelegant (Mohanoor, 2017).

6.3.2.2 Default Kontext

Beim Aufruf des Helper-Bots soll sogleich die E-Mail des Studenten und die Assignment-Nummer an Dialogflow übermittelt werden, sodass diese Werte als Kontext-Variablen verwendet werden können. Wie in Abbildung 10 zu erkennen, kann der Kontext erst auf Ebene der Fulfillments gesetzt werden, wenn die ersten beiden Ebenen nicht kontrolliert werden. Weil aber zu diesem Zeitpunkt die Erkennung der Absicht und des Kontexts durch Dialogflow bereits stattgefunden hat, ist der Kontext erst beim zweiten Aufruf von Dialogflow verfügbar. Besser wäre es, wenn die Kontext-Variablen schon zu Beginn der Konversation übertragen werden und Dialogflow direkt zur Verfügung stünden.

6.3.2.3 Asynchrone Aufrufe

Das grösste Problem stellt die limitierte Bearbeitungszeit bei Fulfillments von 5 Sekunden dar. Während diesen 5 Sekunden können beliebig viele Antworten generiert werden, danach bricht Dialogflow die Verbindung jedoch ab und nimmt keine Antworten mehr entgegen. Da die Code-Bewertung und das Generieren von Feedback zumeist länger als 5 Sekunden benötigt, ist die Hilfestellung über Fulfillments nicht möglich. Die Antworten können wie in Abbildung 10 gezeigt nicht mehr an Dialogflow zurückgespielt werden.

6.3.3 Eigenes Chat-Interface als Lösung

Der Lösung dieser drei Probleme liegt die Überlegung zugrunde, dass mit einem eigenen Chat-Interface und dessen Integration mehr Kontrolle über den gesamten Chat-Bot erlangt werden kann. Einerseits kann dadurch das Persistieren von Konversationen entsprechend Abbildung 10 neu auf Seite der Integration verlagert werden. So wird jede Benutzereingabe direkt persistiert, ohne dass sie zuvor durch Dialogflow verarbeitet wird. Antworten von Dialogflow werden vor dem Anzeigen im Interface gespeichert. Andererseits kann das Setzen des Kontexts bereits auf Ebene der Integration stattfinden, wodurch der Kontext bereits bei der erstmaligen Verarbeitung einer Benutzereingabe bekannt ist.

Ebenfalls zeigt Abbildung 10 die Lösung zur limitierten Bearbeitungszeit. Normalerweise durchläuft die Verarbeitung einer Eingabe alle Schritte und wieder zurück. Um nun die limitierte Bearbeitungszeit zwischen Fulfillment- und Dialogflow-Ebene zu überbrücken, werden asynchrone Antworten nicht über den herkömmlichen Weg via Dialogflow und Integration prozessiert, sondern direkt an das eigene Chat-Interface weitergeleitet. Die Erfüllung des Intents "Hilfe" könnte es nun vorsehen, dass entsprechend Abbildung 10 zuerst eine synchrone Antwort wie "Einen Moment bitte" angezeigt und in der Zwischenzeit der Code überprüft und das Feedback als asynchrone Antwort verarbeitet wird. Die Integrations-Schnittstelle und Code-Beispiele sind im in Tabelle 9 dokumentiert.

Die vier Chatbot-Ebenen finden sich auch in der finalen Systemarchitektur in Abbildung 11 wieder, wobei die Ebenen Integration und Fulfillment als Schnittstellen angezeigt sind.

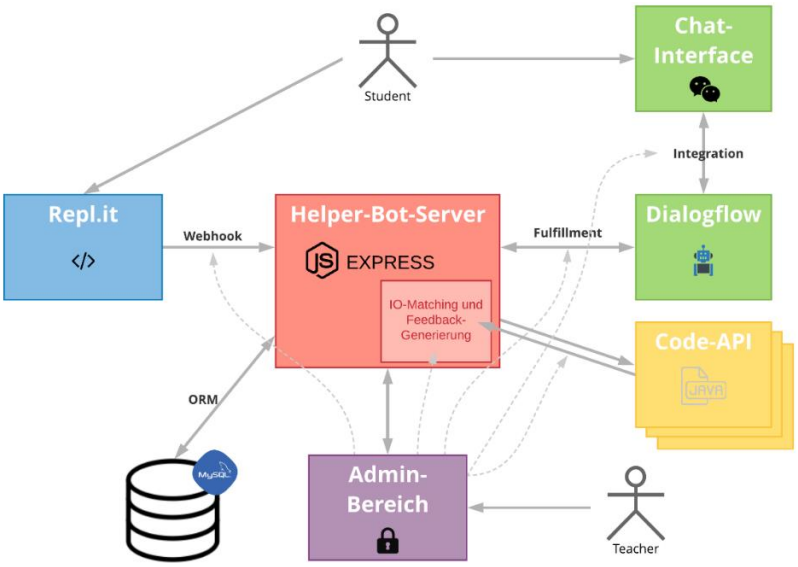


Abbildung 11: Finale Systemarchitektur nach Integration des Chatbots

7 Resultate

Während das vorhergehende Kapitel als Dokumentation der Architektur, Schnittstellen und Tools dient, werden in diesem Kapitel die Resultate des Helper-Bots auf Grundlage der einzelnen Komponente gezeigt und mit den Anforderungen verglichen. Je nach Anforderung und Komponente wird zur Demonstration der Resultate entweder ein grafischer Ausschnitt aus der Anwendung oder Konfigurationseinstellungen gezeigt. An den entsprechenden Stellen werden jeweils die Anforderungen aus dem Anforderungskatalog im Anhang A referenziert.

7.1 Helper-Bot-Server

Da die Anforderungen an den Helper-Bots-Server zumeist architekturbedingter Natur sind, wurden bereits die meisten Aspekte der Anforderungen im vorhergehenden Kapitel beantwortet (**A6**). Der Helper-Bot-Server ist fähig Webhooks von repl.it (**A3** & **A4**) und Dialogflow (**A5**) entgegenzunehmen und entsprechend darauf zu reagieren. Als Teil der Resultate sind dazu die Schnittstellen im Anhang D beschrieben.

Bezüglich der Validität von studentischen Daten wurde auf eine umfängliche Validierung verzichtet. Die Daten werden nicht direkt vom Studenten, sondern von repl.it an den Helper-Bot gesendet und dürften daher bereits einen Validierungsprozess durchlaufen haben.

7.2 Admin-Bereich

Die Resultate des Admin-Bereich setzen sich aus diversen Ansichten für Live-Daten und Konfiguration zusammen. Da es nicht Ziel dieser Arbeit war, ein eigenes Template zu kreieren, wurde stattdessen das statische HTML/CSS-Template `Start Bootstrap - SB Admin 2`[1] als Ausgangslage verwendet, welches unter der MIT-Lizenz frei genutzt werden kann. Um das Template mit dynamischen Inhalten abzufüllen (**A20**), wurde dieses unter Anwendung der Template-Engine EJS angepasst. EJS ist eine einfache Template-Sprache in der Node-Umgebung, mit welcher ein dynamisches HTML-Markup mittels JavaScript generiert werden kann. Abbildung 12 zeigt inwiefern sich das angepasste Template des Admin-Bereichs vom Original unterscheidet.

[1] https://github.com/BlackrockDigital/startbootstrap-sb-admin-2

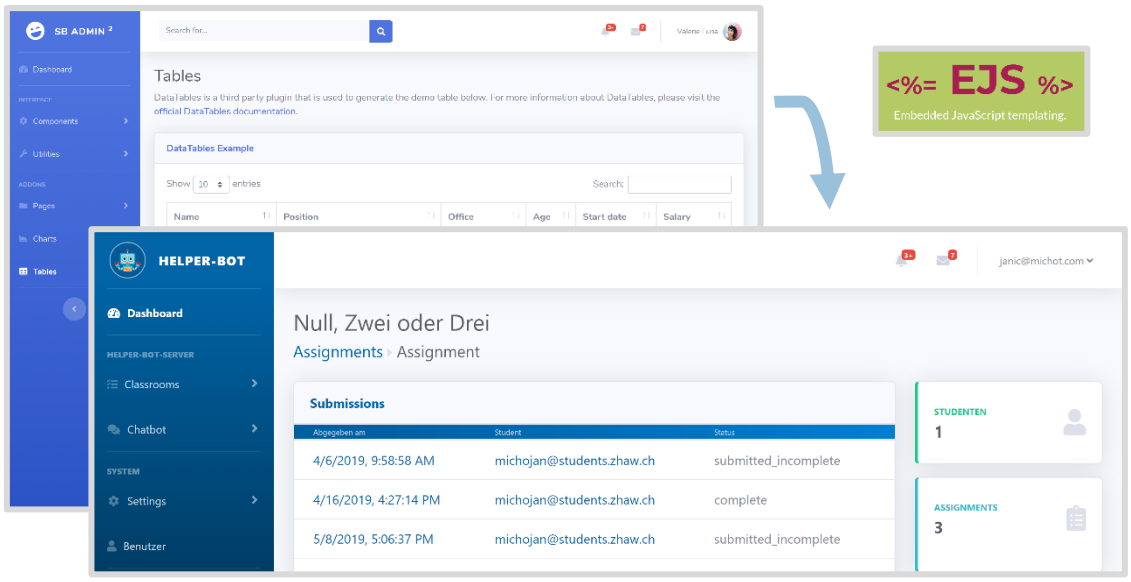


Abbildung 12: Vergleich zwischen originalem und angepasstem Template

7.2.1 Passwortschutz

Nur Dozenten ist der Zugriff auf den Helper-Bot und den studentischen Daten vorbehalten, weshalb der Admin-Bereich passwortgeschützt sein muss (**A19**). Auch aus datenschutztechnischen Gründen wurde dieser Anforderung Sorge getragen.

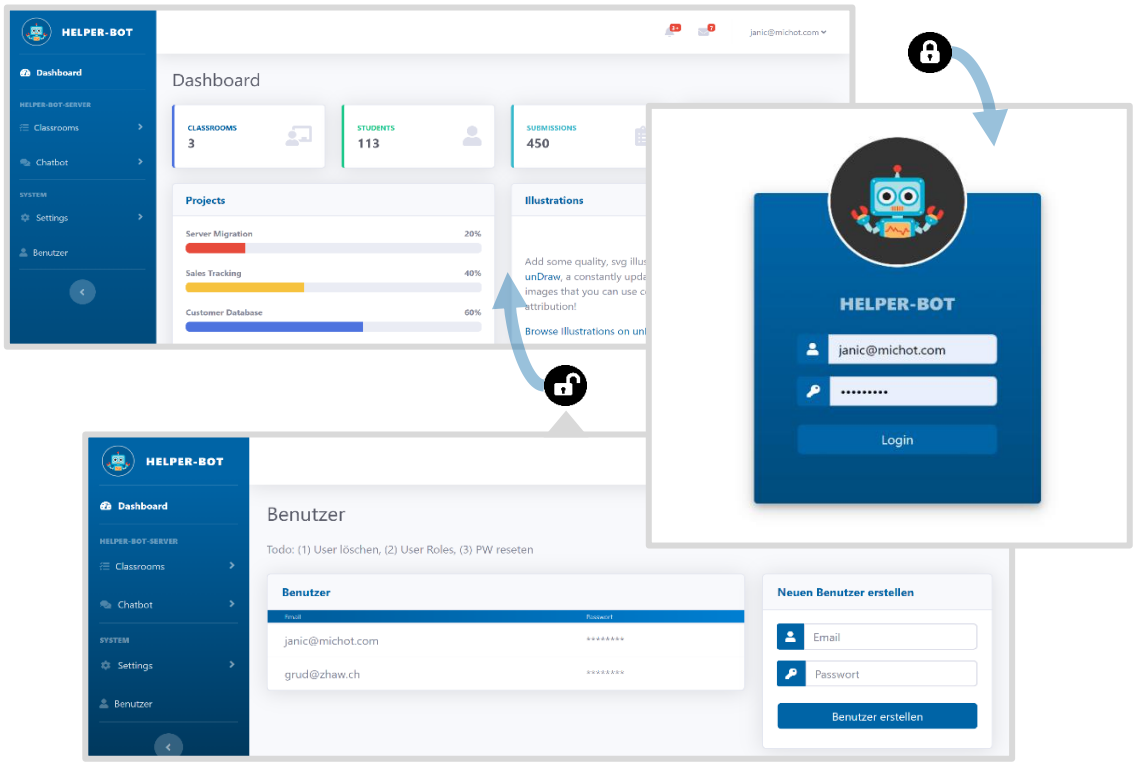


Abbildung 13: Passwortschutz im Admin-Bereich

In Abbildung 13 ist der Passwortschutz grafisch zusammengefasst. Versucht ein nicht-eingeloggter Client auf dem Admin-Bereich zuzugreifen, wird dieser zum Login-Fenster weitergeleitet. Das Login erfolgt jeweils über eine Kombination aus E-Mail und Passwort, wobei neue Benutzer nur über den Admin-Bereich hinzugefügt werden können. Stimmen die Benutzereingaben mit den in der Datenbank hinterlegten Daten überein, wird der autorisierte Benutzer zum Dashboard weitergeleitet.

7.2.2 Live-Daten

Über die Navigation, zu sehen in Abbildung 14, auf der linken Seite können sowohl Live-Daten, wie auch Konfigurationsseiten aufgerufen werden. Für die Live-Daten gibt es unter dem Navigationspunkt `Classroom` die beiden Menüpunkte `Assignments` und `Students`, die zum jeweiligen Inhalt führen.

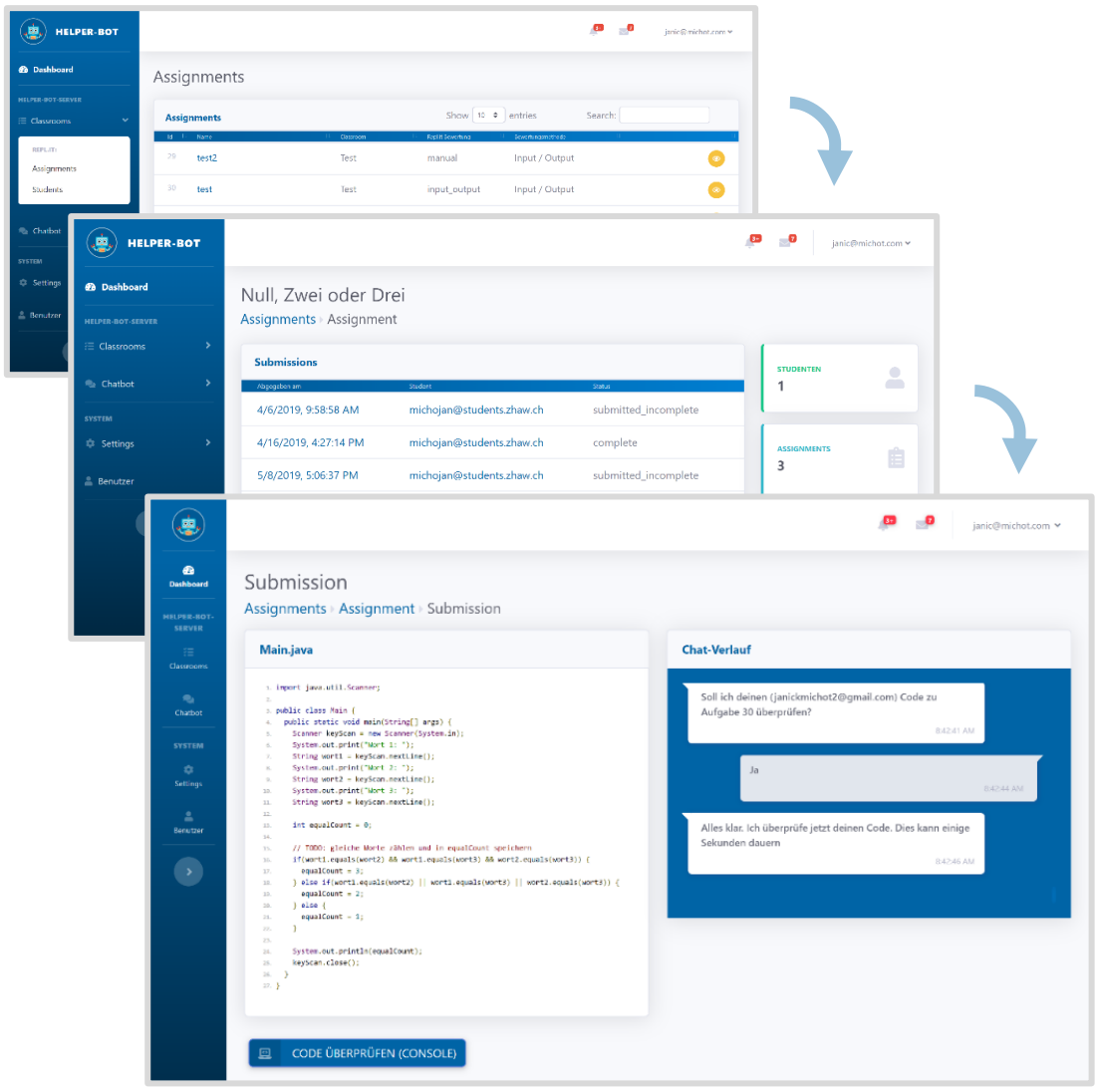


Abbildung 14: Struktur und Darstellung der Live-Daten im Admin-Bereich

Die in Abbildung 14 abgebildete Abfolge von Ansichten gilt nicht nur für die dargestellten `Assignments`, sondern gleichermassen für `Students`. In der ersten Ansicht werden alle Assignments, respektive alle Studenten aufgelistet. Zur erleichterten Bedienung können die angezeigten Inhalte sortiert und durchsucht werden. In der zweiten Ansicht werden alle Submissions zu einem Assignment oder einem Studenten angezeigt. Zuletzt wird in der dritten Ansicht eine solche Submissions inklusive Files und Chat-Verlauf dargestellt.

7.2.3 Konfiguration und Parametrisierung

Für welche Schnittstellen Konfigurations- und Parametrisierungs-Seiten erstellt wurden, ist bereits in der Architekturdokumentation im Kapitel 6 beschrieben und kann aus Abbildung 11 entnommen werden. Dazu zählen Seiten für den repl.it-Webhook, Dialogflow Fulfillments und die Code-Analyse (**A21**).

Die Konfigurationsseite für repl.it-Webhooks ist in Abbildung 15 dargestellt. Diese dient gleichzeitig der Zugriffskontrolle von repl.it-Webhooks und der Autorisierung von Aufrufen des Chat-Interfaces. Auf Seite von repl.it werden nur Webhooks authentifiziert, bei denen Name und Classroom-Secret korrekt sind. Anderseits ist der Zugriff auf das Chatbot-Interface nur dann gestattet, wenn das hinterlegte Moodle-Secret in der URL-Anfrage enthalten ist.

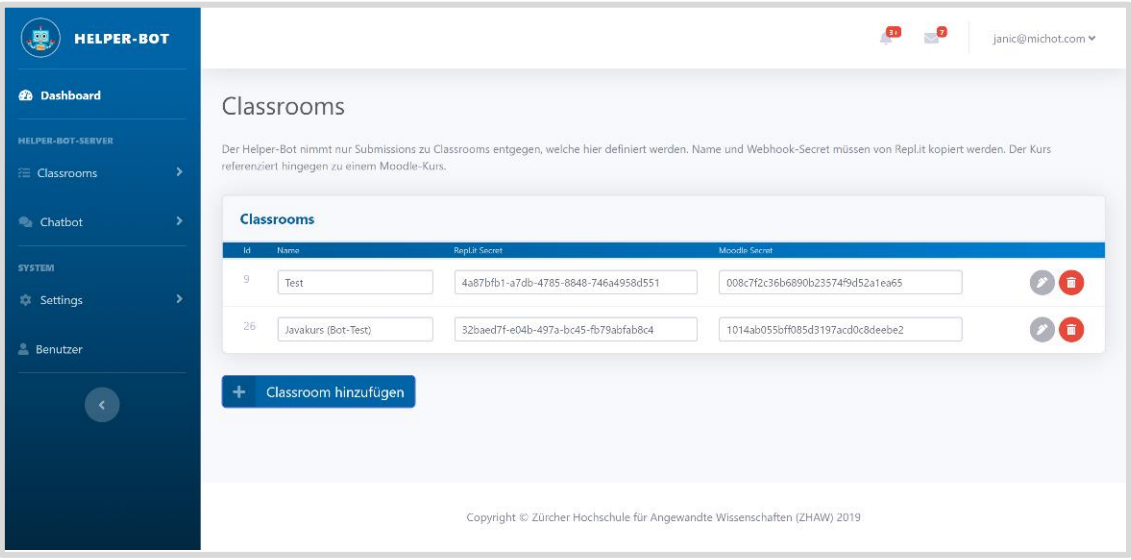


Abbildung 15: Konfigurationsseite für die Authentifizierung von Classrooms

7.3 Code-Analyse

Wohingegen das Zusammenspiel zwischen der isolierten Code-Analyse und dem Helper-Bot in der Architekturdokumentation im Kapitel 6.2 im Vordergrund steht, liegt der Fokus dieses Kapitels auf der Funktionsweise der Code-Analyse. Damit soll die Erfüllung der Anforderungen an das statische (**A15**) und dynamische (**A16**) Assessment sowie an die Feedback-Generierung (**A17**) gezeigt werden.

7.3.1 Sicherheitsanforderungen

Die Trennung zwischen Helper-Bot-Server und Code-API ist primär auf technische Entscheidungen bezüglich des Ausführens von Java-Code zurückzuführen. Gleichwohl ermöglich sie es, den Code in einer gesicherten Umgebung (Sandbox) auszuführen (**A17**). Bösartiger Code kann dem Helper-Bot nicht direkt Schaden anrichten oder unerlaubt auf Daten zugreifen. Unabhängig davon müsste der Code vor jeder Ausführung überprüft werden, sodass auch die Code-API vor bösartigem Code bewahrt wird. Da der Code nicht direkt vom Studenten, sondern von repl.it übermittelt wird und gewisse Sicherheitsaspekte erfüllt sein dürften, wurde für die prototypische Umsetzung auf eine umfängliche Untersuchung nach schädlichem Code verzichtet.

7.3.2 Funktionsweise Code-Überprüfung

Für die Code-Überprüfungs-API wurde eine einfache Spring-Applikation erstellt, welche über einen einzelnen Endpoint angesteuert werden kann. Dieser empfängt die Daten der POST-Anfrage und mappt diese in ein POJO (Plain Old Java Object). Darin enthalten ist anschliessend das studentische Programm sowie die zu überprüfenden Inputs. Die eigentliche Code-Auswertung orientiert sich an der Anleitung von Gerard (2018), worin er beschreibt, wie man Java-Code zur Laufzeit dynamisch erstellt, kompiliert und ausführt. Der darin beschriebene Code wurde weitgehend übernommen und entsprechend der Bedürfnisse des Helper-Bots angepasst.

Die gesamte Funktionsweise der Code-Überprüfung und der Feedback-Generation ist in Abbildung 16 als Flowchart (BPMN) abgebildet. Darin wird nicht nur der Ablauf innerhalb der Code-API gezeigt, sondern auch deren Aufruf. Diesbezüglich findet für jeden definierten Testfall ein eigener API-Aufruf statt, sofern im vorherigen Aufruf kein Kompilierungs-Fehler aufgetreten ist. Denn dieser würde beim Aufrufen des gleichen Codes wieder auftreten.

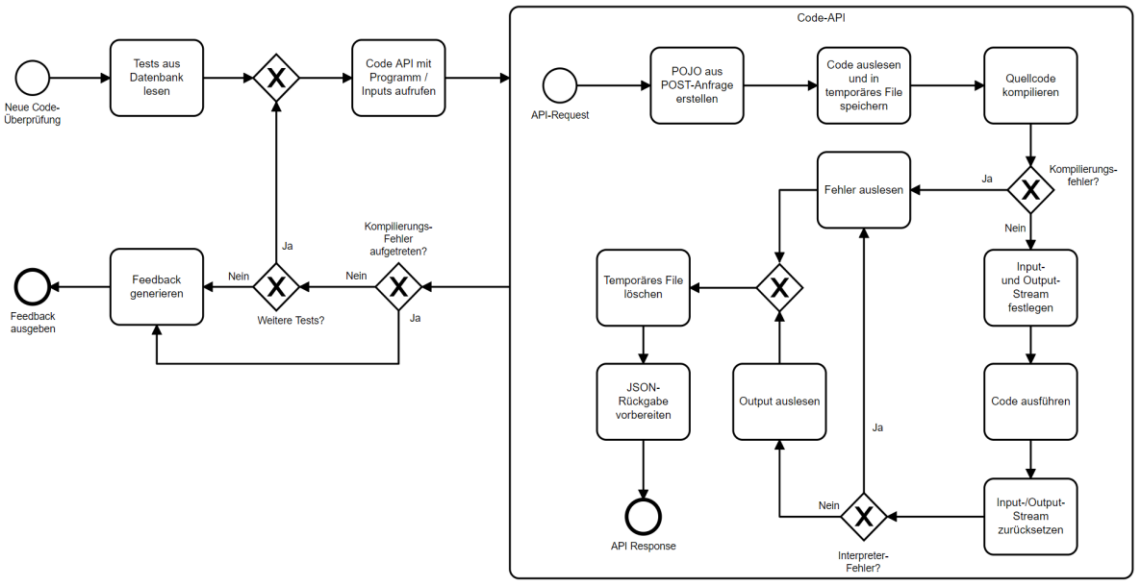


Abbildung 16: Funktionsweise der Code-Überprüfung und Feedback-Generierung

Der Code wird zunächst in ein temporäres File geschrieben, welches anschliessend kompiliert wird. Offenbart das Programm syntaktische Fehler bricht der Compiler ab und der entsprechenden Fehler kann ausgelesen werden. Dabei zitiert der Compiler relativ genau wo der Fehler im Quellcode auftritt, indem er Start- und Endposition angibt (Gerard, 2018). Dadurch kann der Fehler zumeist relativ genau lokalisiert werden. Allfällige Kompilierungs-Fehler werden zurück an den Helper-Bot gesendet.

Nur wenn der Compiler keine Fehler erkennt, wird die studentische Abgabe anschliessend durch den Interpreter ausgeführt. Zuvor muss aber der Input- und Output-Stream umgeleitet werden. Die Ausgabe soll nicht über die Konsole erfolgen und stattdessen in eine Variable geschrieben werden. Bezüglich des Input-Streams muss das Programm nicht auf eine Benutzereingabe über die Konsole warten (Scanner), sondern die per API-Call übergebenen Input-Werte verwenden. Tritt während der Ausführung des Programms ein Fehler auf, wird dieser und ansonsten der Output des Programmes als JSON zurückgegeben. Die Feedback-Generierung findet wiederum zentralisiert auf Seite des Helper-Bots statt.

7.3.3 Statische Code-Überprüfung

Das statische Assessment findet in Abbildung 16 unter dem Punkt `Quellcode kompilieren` statt und beschränkt sich vorerst auf die syntaktische Korrektheit des Programmes. Tritt ein Kompilierungs-Fehler auf, wird dieser an den Helper-Bot zurückgesendet und dort entsprechendes Feedback generiert. Die Parametrisierung von Feedback erfolgt über

den Admin-Bereich. Abbildung 17 zeigt neben der URL-Definition der Code-Analyse, wie Fehlermeldungen festgelegt werden (**A22**).

Tritt bei der Überprüfung ein Fehler auf, für welchen noch kein Feedback hinterlegt wurde, wird stattdessen die originale Java-Fehlermeldung ausgegeben. Gleichzeitig wird die Standard-Fehlerliste um den entsprechenden Fehler ergänzt (mit gelbem Icon markiert). Dadurch ist das Auftreten eines neuen Fehlers für einen Admin ersichtlich und kann entsprechend ergänzt werden. So bildet sich mit der Zeit eine vollständige Liste der meistauftretenden Fehler.

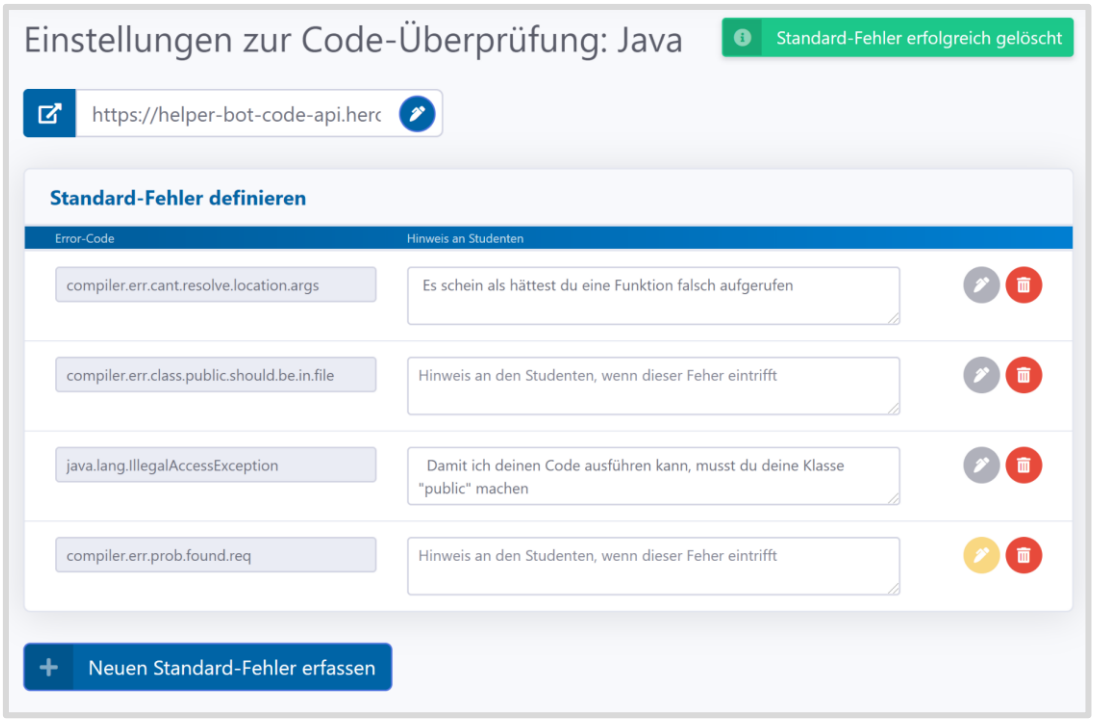


Abbildung 17: Konfigurationsseite für Standardfehler der Code-Überprüfung

7.3.4 Dynamische Code-Überprüfung

Fehler die beim Ausführen des Programmes auftreten, werden ebenfalls über die in Abbildung 17 gezeigte Fehler-Liste behandelt. Für die Beurteilung des Outputs wird entsprechend der Funktionstests nach Caiza und Álamo Ramiro (2013) die Funktionalität des Programmes mittels IO-Matching bewertet. Die Bewertung findet vorerst nur auf System- und nicht auf Methoden-Level statt. Dadurch muss die Analyse nicht auf der Code-API stattfinden und kann stattdessen zentralisiert auf dem Helper erfolgt.

Feedback bezieht sich beim IO-Matching im Gegensatz zu den Java-Fehlern auf ein einzelnes Assignment. Daher findet die Parametrisierung, wie in Abbildung 17 gezeigt, über eine Einzelansicht des jeweiligen Assignments statt (**A22**). Für jedes Assignment können beliebig viele Testregeln geschrieben werden, die bei der Analyse alle ausgeführt werden. Hierbei wird zwischen Tests unterschieden, welche es zu erfüllen gilt und solchen, die nicht eintreffen dürfen. Eine Aufgabe wird nur dann als korrekt gewertet, wenn alle zu erfüllenden Tests korrekt sind und kein falscher Output eintrifft. Das jeweilige semantische Feedback wird entweder dann ausgegeben, wenn ein zu erfüllender Test nicht eintrifft oder dann, wenn ein falscher Test eintrifft. Bezüglich der Bewertung von Programmiernovizen werden hiermit fast alle Möglichkeiten zur Überprüfung einer einfachen Aufgabe abgedeckt.

Nach vergleichbarem Prinzip wie bei den zuvor beschriebenen Standard-Fehlern werden Assignments, zu denen es noch keine Überprüfungsregeln gibt, trotzdem gespeichert und in der Übersicht (siehe Abbildung 18) durch ein rotes Plus hervorgehoben.

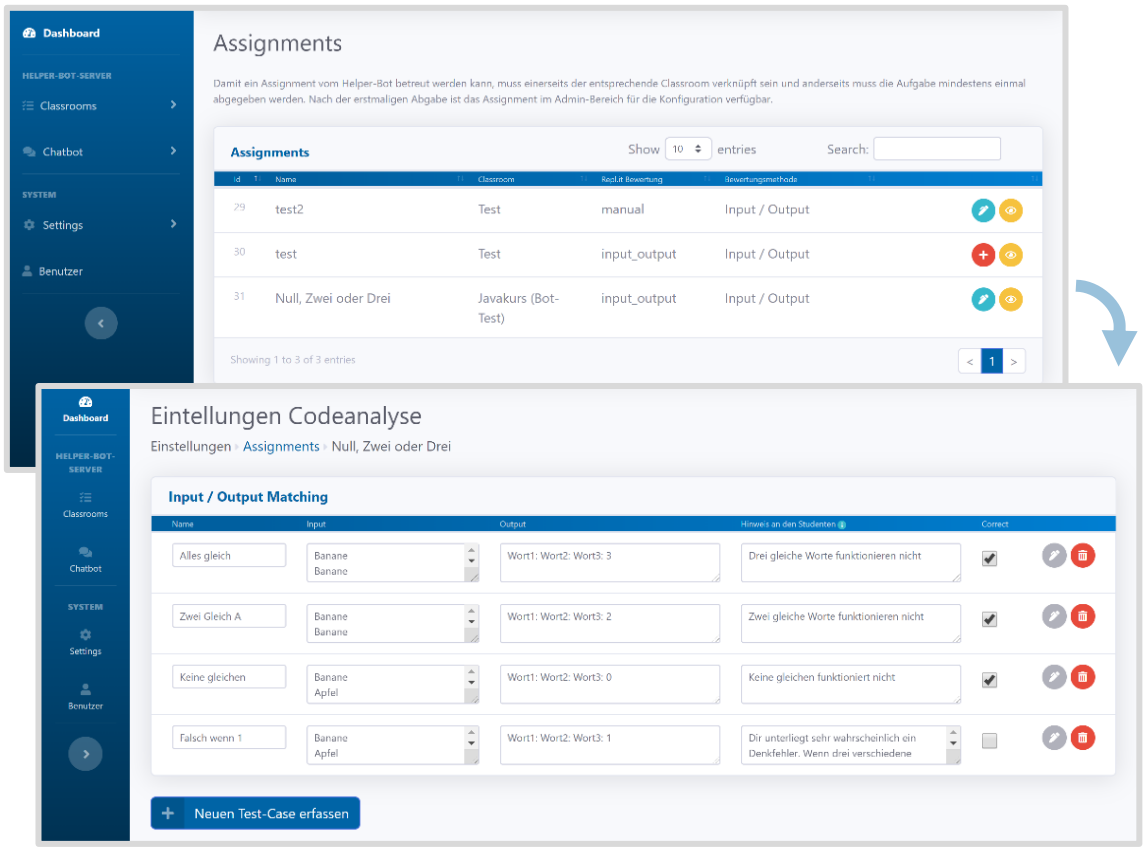


Abbildung 18: Konfigurationsseiten für die dynamische Code-Überprüfung

7.4 Chatbot

Bereits in der Architekturbeschreibung im Kapitel 6.3 wurde der Aufbau eines Dialogflow-Chatbot und die Unterteilung in vier Ebenen erwähnt. Auch die Präsentation der Resultate orientiert sich an dieser Aufteilung und zeigt jeweils die Ergebnisse auf den Ebenen UI, Integration, Konversation und Fulfillment.

7.4.1 UI-Ebene

Als Voraussetzung für die UI-Ebene muss der Chatbot über einen Browser aufrufbar sein. Ausgehend von Moodle wird hiernach gezeigt, wie ein Student auf das Chat-Interface zugreifen kann und welche Informationen dabei übermittelt werden (**A2** & **A13**).

7.4.1.1 URL-Variablen

Hierfür ermöglicht es Moodle-Einstellungen einen Link zu erstellen und diesem URL-Variablen zuzuteilen. Als Teil der URL (Queries) werden diese an die entsprechende Web-Anwendung in Form von Key-Value-Paare weitergeleitet. Für die Values gibt es verschiedene vordefinierte Werte, jedoch keine Möglichkeit eigene zu definieren. Insofern wird zwischen benutzerspezifischen und aufgabenspezifischen Parametern unterschieden. Für benutzerspezifische werden URL-Variablen von Moodle verwendet und für anwendungsspezifische kommen stattdessen dynamische Pfadangaben zur Anwendung. Als benutzerspezifisch gilt die Studenten-E-Mail und als aufgabenspezifisch die Aufgabennummer und das Classroom-Secret:

url/chat/secret_key/31?student=michojan%40students.zhaw.ch

7.4.1.2 Das Chat-Interface

Über diese URL gelangt ein Student sodann auf das in Abbildung 19 gezeigte Chat-Interface des Helper-Bots. Beim Aufruf wird jeweils eine statische Willkommensnachricht angezeigt. Ebenfalls wird der Standard-Intent, Hilfe zu erhalten, zu Beginn bereits ausgelöst. Daher erscheint beim Aufruf des Chat-Interface nach kurzer Verzögerung entweder die Frage, ob der Code überprüft werden soll, oder falls eine Entität fehlt, wird diese per Nachfrage ermittelt. Dadurch wird es dem Studenten zugleich klar, welche Eingaben er tätigen kann und muss (**A10**). Infolgedessen, dass ein eigenes Interface verwendet wird, kann die Darstellung von Inhalten (Rich-Contents) selber bestimmt werden. Vorerst werden aber nur einfache Textnachrichten ausgegeben (**A14**).

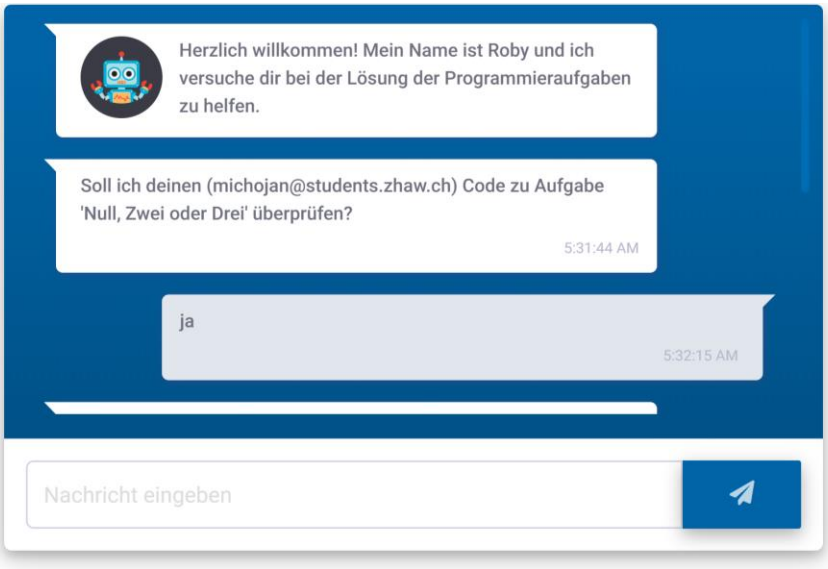


Abbildung 19: Chat-Interface mit Standard-Nachricht und -Intent

7.4.2 Integrations-Ebene

Die Verwendung eines eigenen Chat-Interfaces verlangt deren Integration in Dialogflow (**A13**). Hierfür gibt es den Dialogflow Node.js Client, über welchem Benutzereingaben an die Sprachverarbeitung von Dialogflow gesendet werden können.

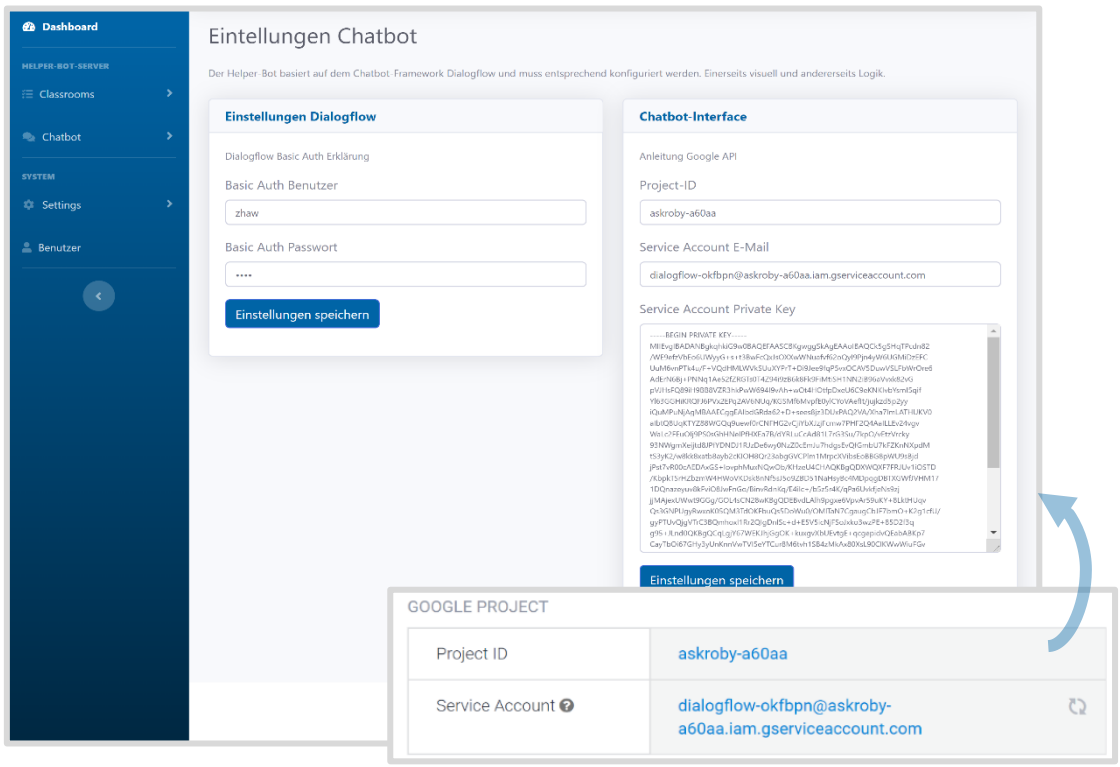


Abbildung 20: Konfigurationsseits für Chatbot-Integration und Fulfillments

Dem Node-Client liegt das Konzept der Dienstkonten zugrunde. Google beschreibt eine Dienstkonto als «ein spezielles Google-Konto, das zu Ihrer Anwendung […] gehört und nicht zu einem bestimmten Endnutzer. Ihre Anwendung ruft mithilfe des Dienstkontos die Google API eines Dienstes auf, sodass die Nutzer nicht direkt beteiligt sind» (Google Cloud, o. J.). Solche Dienstkonten sind mit einem Schlüsselpaar verbunden, die auf der Google Cloud Plattform verwaltet werden können.

Zur vereinfachten Einrichtung der Authentifizierung des Projekts und des Dienstkontos wurde hierfür eine Konfigurationsseite erstellt. Diese ist in Abbildung 20 zusammen mit den Einstellungen auf Dialogflow zu sehen. Wie die Integrationsebene eingerichtet wird, ist in der Installationsanleitung im Anhang E beschrieben. Tabelle 9 im Anhang D beschreibt hingegen die Schnittstelle zwischen dem Interface und Dialogflow und zeigt Code-Beispiele.

7.4.3 Konversationsebene

Die Konversationsebene bezieht sich auf die Einstellungen von Intents und Entitäten in der Dialogflow-Umgebung. Abbildung 21 fasst diese Einstellungen zusammen (**A11**).

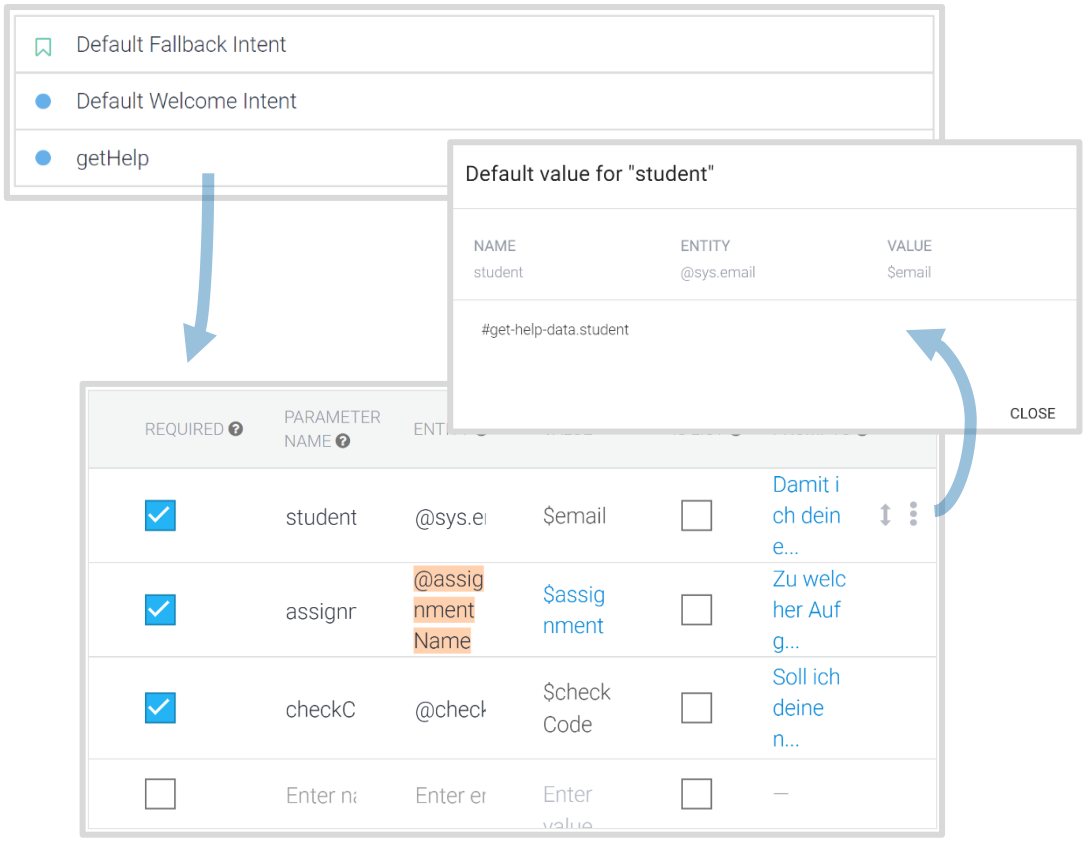


Abbildung 21: Konversationseinstellungen von Intents und Entitäten in Dialogflow

Gründe für die geringe Anzahl an Intents in Abbildung 21 ist einerseits die prototypische Umsetzung und andererseits die Bevorzugung von Slot-Filling gegenüber starrer Dialogbäume. Anstatt dass der Chatbot einer starren Abfolge von Fragen zur Erfassung aller Kontext-Variablen folgt, erlaubt es Slot-Filling diese automatisiert und dynamisch zu erfragen (A11). Die für den Intent `getHelp` definierten Entitäten sind ebenfalls in Abbildung 21 dargestellt. Damit Dialogflow aus dem Kontext (E-Mail und Assignment) entsprechende Entitäten ableitet kann, müssen diese als Default-Value eingetragen werden. Mittels Route kann dabei ein Parameter aus einem Kontext gelesen werden. Neben der E-Mail und dem Assignment wird als dritte Entität eine Bestätigung eingeholt, ob der Code auch tatsächlich überprüft werden soll. Für jeden Intent muss separat angegeben werden, ob ein Webhook ausgelöst werden soll.

Die anderen beiden Intents `Welcome` und `Fallback` werden standardmässig von Dialogflow vordefiniert und werden zur Begrüssung, respektive zur Klärung einer nicht verstandenen Benutzereingabe ausgelöst.

7.4.4 Fulfillment-Ebene

Auch zur Verarbeitung von Fulfillments stellt Dialogflow eine Library zur Verfügung. Mit der `Dialogflow Fulfillment Library` können Webhook-Anfragen entgegengenommen und entsprechende Antworten generiert werden. Entsprechende Code-Beispiele werden in der Schnittstellenbeschreibung in Tabelle 8 gezeigt. Damit nur autorisierte Dialogflow-Anfragen prozessiert werden, wurde Basic-Authentifikation eingerichtet. Dabei wird bei jedem Aufruf Benutzer und Passwort überprüft. Diese beiden Werte müssen sowohl im Admin-Bereich (Abbildung 20 linke Seite) als auch in den Dialogflow Fulfillment-Einstellungen angegeben werden. Wird der Webhook autorisiert, wird anschliessend die entsprechende Funktion aufgerufen.

Als Bestandteil des Ergebnisses dieser Arbeit befinden sich im Anhang C Beschreibungen zur Anwendungsstruktur, der verwendeten Libraries sowie das komplettes Datenbankschema des Helper-Bots. Im Anhang D werden sämtliche Schnittstellen dokumentiert und entsprechende Code-Ausschnitte gezeigt. Weiter wird im Anhang E beschrieben, wie der Helper-Bot installiert werden kann.

8 Fazit

In diesem abschliessenden Kapitel wird zunächst der Helper-Bot evaluiert und aus den Ergebnissen Implikationen abgeleitet. Ziel der Evaluation ist die Reproduzierbarkeit und Nachvollziehbarkeit, sodass ein unabhängiger Untersucher dieselben Resultate provozieren könnte. In den Implikationen werden die Ergebnisse zur Beantwortung der Forschungsfrage interpretiert. Abgerundet wird dieses Kapitel, respektive diese Thesis durch eine Schlussfolgerung.

8.1 Tests und Evaluation

Als Ausgangslage für diese Evaluation dient eine bereits existierende Programmieraufgabe, die als iFrame in eine Moodle-Kursseite implementiert ist. Ebenfalls wird für den Test vorausgesetzt, dass der Helper-Bot entsprechend der Installationsanleitung im Anhang E erfolgreich installiert wurde und dass der entsprechende Classroom bereits im Admin-Bereich konfiguriert ist. Die letztgenannte Voraussetzung kann gegebenenfalls auch als Bestanteil der Evaluation berücksichtigt werden. Die Evaluierung sieht folgende Abfolge von Schritten vor:

1) Tests und Feedback für eine Programmieraufgabe definieren oder anpassen (Assignments sind momentan nur nach erstmaliger Submission verfügbar. Falls Aufgabe noch nicht im Admin-Bereich verfügbar ist, zuerst einen Webhook über repl.it auslösen, beziehungsweise Aufgabe abgeben)
2) Programmieraufgabe über Moodle lösen, respektive absichtlich Fehler (logische oder syntaktische) provozieren
3) Programm zur Überprüfung abgeben (im Admin-Bereich überprüfen, ob Submission erfolgreich an den Helper-Bot übermittelt wurde)
4) Chatbot aufrufen und Hilfe anfordern
5) Feedback überprüfen
6) Schritte 2 – 5 mit anderen Fehlern, respektive mit der korrekten Lösung wiederholen

Die Abfolge von Schritten ist so gewählt, dass sämtliche Komponenten des Helper-Bots als Teil der Evaluation berücksichtig werden. Entsprechend der Definition von Testen im Bereich der Softwareentwicklung nach Myers, Badgett, Thomas und Sandler (2004) als Prozess, ein Programm mit der Absicht auszuführen, Fehler zu finden, soll mit dieser Abfolge von Schritten allfällige Fehler aufgedeckt werden. Da sich die Fehler in ihrem Ursprung unterscheiden, muss die Anwendung aus verschiedenen Perspektiven evaluiert werden. Mit der obigen Schrittabfolge wird der Helper-Bot zwar auf Systemebene untersucht, jedoch werden gleichzeitig die einzelnen Komponenten sowie die Integration von Umsystemen miteinbezogen.

Um die Funktionalität des Helper-Bots zu beurteilen, ist in Tabelle 12 im Anhang F zu jedem Schritt die jeweils betroffenen Komponenten und deren Aufgabe aufgeführt. Wenn alle Fragen dieser Tabelle mit ja beantwortet werden können, war der Test erfolgreich. Bezüglich Schritt 5 wird in der separaten Tabelle 13 im Anhang F der jeweilige Code, mit den definierten Tests verglichen und untersucht, ob das erwartete Feedback bei einer Hilfestellung mit dem tatsächlichen übereinstimmt.

Im Rahmen der Evaluation wurde dieser Test drei Mal wiederholt: Einmal mit einem syntaktischen Fehler, einmal mit einem Output-Fehler und ein letztes Mal mit einer richtigen Lösung. Es sind dabei keine Fehler aufgetreten und die erwarteten Rückmeldungen haben jeweils mit der tatsächlichen übereingestimmt. Bei syntaktischen Fehlern die nicht über den Admin-Bereich definiert wurden, wird der standardmässige Java-Fehler ausgegeben. Zuvor unbekannte Fehler sollten jedoch automatisch in die Fehler-Liste im Admin-Bereich aufgenommen werden. Aus funktionaler Sicht hat der Test gezeigt, dass der Helper-Bot fähig ist, auf Benutzereingaben zu reagieren, diese zu prozessieren und entsprechendes Feedback zu generieren. Wenngleich der Test erfolgreich war, besitzt die Anwendung mit den vielen einzelnen Komponenten und Schnittstellen grosses Fehlerpotential. Bis zum praktischen Einsatz müssen daher weiter Tests, insbesondere Usability-Tests mit Studenten, durchgeführt werden.

Der didaktische Nutzen und die qualitative Verbesserung eines Kurses können mit diesem Test hingegen nicht evaluiert werden. Inwiefern der Helper-Bot zum Lernerfolg der Studenten beiträgt, wird sich nach erstmaligem akademischem Einsatz zeigen. Gleiches gilt für eine potentielle Einsparung von menschlichen Ressourcen. Um die Benutzerakzep-

tanz zu bemessen, wäre es beispielsweise denkbar am Ende einer Konversation den Studenten nach seiner Zufriedenheit zu befragen. Die Zufriedenheit könnte ein Student sowohl in Form einer Likert-Scala oder als schriftliche Verbesserungsvorschläge ausdrücken. Die Akzeptanz könnte sodann als Indikator für den Erfolg des Helper-Bots fungieren.

8.2 Implikationen

Das Ergebnis dieser Arbeit hat gezeigt, dass der Helper-Bot in die bestehende Umgebung implementiert werden kann. Unter Anwendung der NLU-Plattform Dialogflow und der Entwicklung weiterer Komponenten ist die Realisierung einer Anwendung möglich, welche einem Studenten bei Bedarf Hilfe zur Lösung einer Programmieraufgabe leistet. Damit knüpft der Helper-Bot an den, im Forschungsstand untersuchten, Systemen an und ergänzt diese um die Vorteile der Chatbot-Technologie. Mit dem Artefakt dieser Arbeit wurde jedoch kein fertiges Produkt, sondern eine Ausgangslage für eine umfassende Anwendung geschaffen. Bis zum praktischen Einsatz gilt es noch einige didaktische, technische und sicherheitsbedingte Herausforderungen zu bewältigen. Nachfolgend wird die Erfüllung der Forschungsfragen untersucht und daraus Implikationen für die praktische Anwendung des Helper-Bots abgeleitet.

Diverse Chatbot-Plattformen erlauben es mit einfachen Mitteln einen Chatbot zu entwickeln. Rein durch das Parametrisieren von Absichten und Entitäten kann so ein einfacher Chatbot realisiert werden. Da aber die Funktionalitäten des Helper-Bots weit über Question & Answer hinausgehen, musste der Chatbot per Anbindung eines externen Service mit eigener Logik ausgestattet werden. Insbesondere Dialogflow und die Unterstützung von Webhooks machen es möglich, den Helper-Bot um anwendungsspezifische Fähigkeiten zu ergänzen. Vorerst beschränken sich die Fähigkeiten des prototypischen Helper-Bots auf die Hilfestellung. Jedoch wurde darauf geachtet, dass sich der Helper-Bot problemlos um weitere Fähigkeiten ergänzen lässt. Für die praktische Nutzung müssten unter Berücksichtigung des didaktischen Nutzens weitere Intents erfasst und wenn nötig, entsprechende Fulfillments entwickelt werden.

Nicht nur bei der Gestaltung der Konversation und der Definition von Absichten muss der didaktische Nutzen berücksichtig werden, auch die Formulierung von Feedback trägt zum Erfolg der Anwendung bei. Der Helper-Bot ermöglich es zwar über den Admin-

Bereich Fehlermeldungen für syntaktische und logische Fehler festzulegen, jedoch beschränkt sich die Hilfestellung auf jeweils ein Feedback pro Fehler. Wenn nun dieses Feedback dem Studenten nicht zur Lösung verhilft, ist die Hilfestellung durch den Helper-Bot gescheitert. Wird die Hilfestellung hingegen zu genau formuliert, nimmt sie eventuell die Lösung vorweg. Die Formulierung einer didaktisch sinnvollen Rückmeldung zeigt sich daher als Gratwanderung zwischen genügend hilfreich und zu detailliert.

Eine denkbare Lösung wäre die Definition von mehreren Fehlermeldungen mit unterschiedlichem Detailierungsgrad. Anstatt dass es jeweils nur ein Feedback pro Aufgabe gibt, könnten stattdessen drei Hilfestellungen formuliert werden. Hilft ein Antwort dem Studenten nicht weiter, schickt ihm der Helper-Bot das nächstdetaillierte Feedback. Somit verbessert sich die Chance, dass der Student zur richtigen Lösung kommt, ohne ihm die Lösung vorweg zu nehmen.

Alternativ, respektive zusätzlich dazu könnte die Anwendung mit der Möglichkeit ausgestattet werden, dass wenn die Hilfestellung durch den Helper-Bot nicht zur richtigen Lösung verhilft, ein Dozent in das Gespräch eingeschalten werden kann. Entsprechend der drei manuellen Bewertungsmöglichkeiten nach Ihantola et al. (2010) würde dies einer Kombination aus automatischer und manueller Bewertung entsprechen. Dazu müsste der Chatbot um weitere Fulfillments ergänzt werden, mit welchen ein Dozent in irgendeiner Form kontaktiert werden kann.

Ebenfalls sollte für eine detaillierte Formulierung von Hilfestellungen Platzhalter hinzugefügt werden. Möglich Platzhalter sind beispielsweise Zeile des aufgetretenen Fehlers oder Position des Fehlers. Mit diesen Angaben als Teil des Feedbacks kann der Student den Fehler anschliessend genauer lokalisieren. Im momentanen Code-Assessment ergeben sich aber nur Platzhalter für syntaktisches und nicht für semantisches Feedback. Auch wenn die Möglichkeiten zur Feedback-Formulierung weiter verbessert werden kann, hängt deren Qualität schlussendlich von der Formulierung einer Lehrperson ab.

Hinsichtlich der Implementation in die bestehende Umgebung und der Wahl einer geeigneten Systemarchitektur, waren Überlegungen zu denjenigen Informationen zentral, die dem Helper-Bot zur Erfüllung einer Hilfestellung verfügbar sein müssen. Dazu zählt der studentische Code, das Ergebnis einer Prüfung des Codes sowie entsprechende Formulierungen von Hilfe zum Output oder zu Fehler des Programmes. Bezüglich der Verfügbarkeit von studentischen Lösungen bietet repl.it zurzeit lediglich einen Webhook an, der

dann ausgelöst wird, wenn der Student eine Lösung zur Überprüfung abgibt. Da zu diesem Zeitpunkt bereits sämtliche Tests durch repl.it stattgefunden haben und der Student den Code erst nach dreimaliger Bestätigung abgeben kann, ist das Vorgehen zur Überprüfung eines Codes nicht wirklich intuitiv. Falls repl.it künftig andere Webhooks, beispielsweise beim Ausführen des Codes, anbietet, wäre es sinnvoll auf einen solchen umzusteigen. Ebenfalls denkbar wäre es, repl.it als IDE zu ersetzen und eine eigene Programmierumgebung in den Helper-Bot zu implementieren. Damit wäre das Zusammenspiel zwischen Helper-Bot und Programmierumgebung noch enger miteinander verbunden. Neben adaptivem Feedback könnte der Helper-Bot dadurch beispielsweise auch vermehrt summatives Feedback geben und so den Studenten motivieren und in seinem Lernprozess fördern.

Der Helper-Bot läuft in einer Node-Umgebung und unter Einsatz der Express-Library. Da zur dynamischen Code-Analyse eine Java-Laufzeitumgebung benötigt wird, wurde die Code-Analyse auf eine separate API verlagert. Obschon diese Lösungsvariante Vorteile bezüglich der Sicherheit und Einfachheit bietet, scheinen Docker für diesen Einsatz besser geeignet zu sein.

Weiter wurde die statische und dynamische Code-Analyse relativ simpel gehalten. So findet die Analyse neben der syntaktischen Überprüfung nur auf Basis des Outputs des Gesamtprogramms statt. Mittels Blackbox-Tests wird der Output nach zuvor festgelegten Input-Werten untersucht und so eine Aussage darüber gemacht, ob der Code valide ist oder nicht. Wenngleich das IO-Matching für die Bewertung von Aufgaben von Programmiernovizen in dem meisten Fällen ausreichen dürfte, könnte die Überprüfung mit weiteren Tests ergänzt werden. Im Forschungsstand wurden diesbezüglich Ansätze und Systeme untersucht, mit welchen der Helper-Bot ausgerüstet werden könnte.

Dem Aspekt der Sicherheit vor schädlichem Code wurde insofern Sorge getragen, dass die Ausführung auf einem isolierten Server stattfindet und dem Helper-Bot dadurch nicht direkt Schaden zugefügt werden kann. Ebenfalls ist zu vermuten, dass auf Seite von repl.it bereits eine Untersuchung stattfindet. Gleichwohl muss der Code für die praktische Nutzung weiteren Sicherheitsüberprüfungen unterliegen und bösartiger Code vor der Ausführung identifiziert werden. Eine Möglichkeit wäre beispielsweise, den Einsatz von fremden Libraries einzuschränken.

Über den Admin-Bereich können Schnittstellen zwischen den einzelnen Teil- und Umsystemen konfiguriert und die Formulierung von Feedback parametrisiert werden. Auch beim Admin-Bereich wurde darauf geachtet, dass dieser inkrementell ausgebaut werden kann. Die einzelnen Ansichten zu Live-Daten sind momentan eher exemplarisch und können künftig mit weiteren Elementen ausgebaut und sinnvoll strukturiert werden. Dank dem Passwortschutz haben zudem nur Dozenten Zugriff auf die sensiblen Studentendaten.

8.3 Schlussfolgerung

Mit der Entwicklung des Helper-Bots wurde keine neuartige Anwendung zur Lernunterstützung entwickelt, sondern eine neuartige Möglichkeit mit einem solchen zu interagieren. Auch wenn der Stand der Forschung gezeigt hat, dass Tools in der akademischen Programmierausbildung Fortschritte gemacht haben, wurde bislang kein auf der Chatbot-Technologie basierender Ansatz gefunden. Wenngleich es bis zum praktischen Einsatz noch einiger Verbesserungen bedarf, wurde in dieser Arbeit ein möglicher Chatbot-Ansatz entwickelt.

Das Ergebnis und die Evaluation dieser Arbeit haben gezeigt, dass der Helper-Bot einem Studenten umgehend Hilfe leisten kann. Die Qualität der Hilfestellungen hängt dabei von der Parametrisierung über den Admin-Bereich ab. Ohnehin dürfte die Formulierung von qualitativem Feedback einen längerfristigen Prozess darstellen. Aufgaben- und Hilfestellung müssen permanent verbessert und aufeinander abgestimmt werden. Semantisches Feedback, das sich nur auf eine Aufgabe bezieht, kann hierbei einfacher konzipiert werden. Syntaktische Fehler unterscheiden sich hingegen in ihrem Ursprung und können nicht gleichermassen vordefiniert werden. Daher empfiehlt es sich vor der Inbetriebnahme des Helper-Bots beispielhafte Lernlösungen durch den Helper-Bot überprüfen zu lassen und so die häufigsten Programmierfehler zu ermitteln. Nur wenn der Helper-Bot hilfreiches Feedback generiert, führt die verkürzte Latenz zwischen Anfrage und Hilfestellung zu einer qualitativen Verbesserung eines Kurses.

Inwiefern mit dem Helper-Bot menschliche Ressourcen eingespart werden können, bleibt fraglich. Zunächst wird der Unterhalt des Helper-Bots sowie die Formulierung von Feedback einen Mehraufwand für die Dozenten bedeuten. Wenn sich jedoch über mehrere

Semester hinweg eine umfassende Sammlung an Feedback zusammengetragen hat und das Zusammenspiel zwischen Programmieraufgaben und Tests funktioniert, scheint es durchaus möglich, dass die Betreuungszeit reduziert werden kann. Die Anwendung mag zwar Dozenten entlasten, gleichwohl kann die menschliche Betreuung nicht gänzlich abgeschafft werden. Die verfügbare Zeit der Dozenten kann dafür für wichtigere Aspekte wie die Betreuung von schwächeren Studenten eingesetzt werden.

Schlussendlich wird der Erfolg einer solchen Anwendung von der Akzeptanz der Studenten abhängen. So hat die Thesis ergeben, dass sich eine Chatbot-Anwendung zur Lernunterstützung aus technischer Sicht realisieren lässt. Ob der Helper-Bot jedoch zu einer qualitativen Verbesserung in der akademischen Programmierausbildung führt, wird sich erst nach erstmaligem Einsatz zeigen.

Literatur- und Quellenverzeichnis

Ala-Mutka, K. M. (2005). A survey of automated assessment approaches for programming assignments. *Computer science education*, 15(2), 83–102.

Ashwini, A. (2017). Should You Use NoSQL Or SQL Db Or Both? [Blog-Beitrag]. Abgerufen von https://medium.com/swlh/should-you-use-nosql-or-sql-db-or-both-349cb26c9add

Bauer-Messmer, B., Fässler, L. & Wyss, M. (2007). *Einführungskurse ins Programmieren: Eine didaktische Herausforderung*. ETH Zurich.

Bennedsen, J. & Caspersen, M. (2007). Failure rates in introductory programming. *SIGCSE Bulletin*, 39, 32–36. https://doi.org/10.1145/1272848.1272879

Bharathi, V. (2019). Compilex is a node.js library which is used to build online code compiler/interpreter websites and webservices [GitHub-Repositroy]. Abgerufen von https://github.com/scriptnull/compilex

Boud, D. & Molloy, E. (2013). *Feedback in higher and professional education: understanding it and doing it well*. Abingdon: Routledge.

Caiza, J. C. & Álamo Ramiro, J. M. del. (2013). Programming assignments automatic grading: review of tools and implementations.

Canonico, M. & De Russis, L. (2018). A comparison and critique of natural language understanding tools. *Cloud Computing*, S. 110–115.

Christensson, P. (2015). Iframe Definition. Abgerufen von https://techterms.com/

Debrah, R. (2018). Building a NodeJS Web App Using PassportJS for Authentication. The DEV Community [Blog-Beitrag]. Abgerufen von https://dev.to/gm456742/building-a-nodejs-web-app-using-passportjs-for-authentication-3ge2

Dialogflow. (o. J.). Dialogflow Documentations. Abgerufen von https://dialogflow.com/docs

Douce, C., Livingstone, D. & Orwell, J. (2005). Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing* (JERIC), 5(3), 4.

Eicher, D. (2016). Chatbots - Definition. Abgerufen von http://www.digitalwiki.de/

Gerard, N. (2018). Java - Dynamic Java code execution (at runtime). Abgerufen von https://gerardnico.com/lang/java/dynamic

Google Cloud. (o. J.). Dokumentation zu Cloud Identity and Access Management. Abgerufen von https://cloud.google.com/iam/docs/service-accounts

HackerEarth. (o. J.). HackerEarth API v3 - Developers Wiki. Abgerufen von https://www.hackerearth.com/docs/wiki/developers/v3/

Hahn, E. (2016). *Express in Action: Writing, building, and testing Node. js applications*. New York: Manning Publications.

Heroku. (o. J.). What is Heroku. Abgerufen von https://www.heroku.com/what

Hunter II, T. (2017). HTTP API Design. In T. Hunter II (Hrsg.), *Advanced Microservices : A Hands-on Approach to Microservice Infrastructure and Tooling* (S. 13–54). Berkeley, CA: Apress.

Ihantola, P., Ahoniemi, T., Karavirta, V. & Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. *Proceedings of the 10th Koli calling international conference on computing education research* (S. 86–93). ACM.

Jensen, N. (2017). Automatisierte Bewertung in der Programmierausbildung – Ausblick. In: O. Bott, P. Fricke, U. Priss & M. Striewe (Hrsg.): *Automatisierte Bewertung in der Programmierausbildung*. S. 393-405. Münster: Waxmann Verlag.

Kelleher, C. & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, 37(2), 83–137.

Keuning, H., Jeuring, J. & Heeren, B. (2016). *Towards a systematic review of automated feedback generation for programming exercises*. Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (S. 41–46).

Langer, P. (2018). *Entwicklung und Evaluierung eines Chatbots am Beispiel der Studienberatung der HAW Hamburg* (Bachelor Thesis). Abgerufen von http://edoc.sub.uni-hamburg.de/haw/volltexte/2018/4364/pdf/PaLanger_BA.pdf

Le, N.-T. (2016). A classification of adaptive feedback in educational systems for programming. *Systems*, 4(2), 22.

Liddy, E. D. (2001). *Natural language processing*. Syracuse University. Abgerufen von https://surface.syr.edu/cgi/viewcontent.cgi?referer=https://scholar.google.ch/&httpsredir=1&article=1019&context=cnlp

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D. et al. (2001). *A multi-national, multi-institutional study of assessment of programming skills of first-year CS students*. Working group reports from ITiCSE on Innovation and technology in computer science education (S. 125–180). ACM.

Mohanoor, A. (2017). The different types of DialogFlow (API.AI) slot filling tasks. Abgerufen von https://miningbusinessdata.com/different-types-api-ai-slot-filling-tasks/

Mohanoor, A. (2019, April 23). The 4 layers of a Dialogflow bot. Abgerufen von https://miningbusinessdata.com/the-4-layers-of-a-dialogflow-bot/

Myers, G. J., Badgett, T., Thomas, T. M. & Sandler, C. (2004). *The art of software testing* (Band 2). Wiley Online Library.

Nguyen, A., Piech, C., Huang, J. & Guibas, L. (2014). *Codewebs: scalable homework search for massive open online programming courses.* Proceedings of the 23rd international conference on World wide web (S. 491–502). ACM.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J. et al. (2007). *A survey of literature on the teaching of introductory programming.* ACM sigcse bulletin (Band 39, S. 204–223). ACM.

repl.it. (o. J.). Documentation for Repl.it. Abgerufen von https://repl.it/site/docs/

Rivero, J. M., Grigera, J., Rossi, G., Robles Luna, E. & Koch, N. (2012). Towards Agile Model-Driven Web Engineering. In S. Nurcan (Hrsg.), *IS Olympics: Information Systems in a Diverse World* (S. 142–155). Berlin, Heidelberg: Springer.

Sandoval, K. (2016, Juni 2). What is the Difference Between an API and an SDK? [Blog-Beitrag]. Abgerufen von https://nordicapis.com/what-is-the-difference-between-an-api-and-an-sdk/

Sequelize. (o. J.). Getting started. Abgerufen von http://docs.sequelizejs.com/

Shevat, A. (2017). *Designing bots: Creating conversational experiences*. Kalifornien: O'Reilly Media, Inc.

Strickroth, S. & Pinkwart, N. (2017). Automatisierte Bewertung in der Programmierausbildung – Eine Übersicht. In: O. Bott, P. Fricke, U. Priss & M. Striewe (Hrsg.):

Automatisierte Bewertung in der Programmierausbildung. S. 393-405. Münster: Waxmann Verlag.

Striewe, M. & Garmann, R. (2017). Automatisierte Bewertung in der objektorientierten Programmierausbildung am Beispiel von Java. In: O. Bott, P. Fricke, U. Priss & M. Striewe (Hrsg.): *Automatisierte Bewertung in der Programmierausbildung*. S. 393-405. Münster: Waxmann Verlag.

Tilkov, S. & Vinoski, S. (2010). Node.js: Using JavaScript to Build High-Performance Network Programs. *IEEE Internet Computing*, 14(6), 80–83. https://doi.org/10.1109/MIC.2010.145

Wieczorrek, H. W. & Mertens, P. (2008). Vorgehen in IT-Projekten. *Management von IT-Projekten* (S. 53–87). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-85291-9_4

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A*: Mathematical, Physical and Engineering Sciences, 366(1881), 3717–3725.

Anhang A: Anforderungskatalog

Nachfolgend werden die Anforderungen an den Helper-Bot tabellarisch und nach den einzelnen Komponenten gegliedert dargestellt. Jede Anforderung besitzt eine eindeutige und fortlaufende Nummer.

Tabelle 2: Anforderungskatalog zur Umsetzung des Helper-Bots

Anforderungskatalog auf Ebene der einzelnen Komponenten	
Komp.	Anforderung
Moodle	**(A1)** Eine Programmieraufgabe muss als iFrame in einen Moodle-Kurs eingesetzt werden
	(A2) Auf einer Kursseite muss es eine Möglichkeit geben, den Helper-Bot aufzurufen und Meta-Daten wie Name und Assignment zu übermitteln
Repl.it	**(A3)** Repl.it muss nach Lösungseinreichung, die Submission an den Helper-Bot senden
Helper-Bot-Server	**(A4)** Der Helper-Bot-Server muss Submissions entgegennehmen und die Daten persistieren
	(A5) Der Helper-Bot-Server muss Anfragen der NLU-Plattform verarbeiten und entsprechende Reaktionen (z.B. Feedback zu Code) erzeugen
	(A6) Der Helper-Bot-Server muss eine modulare Systemarchitektur für die Implementation aller Komponenten besitzen
NLU-Plattform	**(A7)** Die NLU-Plattform muss natürliche Sprachverarbeitung ermöglichen, um Absichten und Entitäten in Benutzereingabe zu identifizieren
	(A8) Da NLU-Plattform muss Deutsch als Sprache unterstützen
	(A9) Die Plattform muss es erlauben einen externen Service aufzurufen, um die eigene Logik zu implementieren
NLU-Einstellungen	**(A10)** Die Plattform muss so parametrisiert sein, dass es für einen Studenten klar ist, wie er sein Anliegen formulieren muss
	(A11) Der Aufbau der Konversation muss so gestaltet sein, dass alle Entitäten (Student, Aufgabennummer usw.) während der Konversation gesammelt werden
Interface	**(A12)** Das Interface muss die Meta-Daten von Moodle auslesen und verarbeiten können
	(A13) Das Interface muss mit der NLU-Plattform interagieren können

Chat-	**(A14)** Das Interface kann Rich-Contents anzeigen (UI-Element, Buttons usw.)
Code-Analyse (Sandbox/Analyzer)	**(A15)** Die Codeanalyse muss ein Programm zur Überprüfung der syntaktischen Korrektheit statisch überprüfen können
	(A16) Die Codeanalyse muss ein Programm dynamisch überprüfen können, indem es auf Programm- oder Methodenebene den Output mit zuvor festgelegten Inputs abgleicht (IO-Matching, respektive Unit-Tests)
	(A17) Die Codeanalyse muss in einer geschützten Umgebung (Sandbox) erfolgen, sodass bösartiger Code kein Schaden anrichtet
	(A18) Entsprechend identifizierter Fehler muss didaktisch sinnvolles Feedback generiert werden
Admin-Bereich	**(A19)** Da der Admin-Bereich ausschliesslich für Dozenten bestimmt ist und aus Gründen des Datenschutzes, muss dieser passwortgeschützt sein
	(A20) Der Admin-Bereich muss dynamische Inhalte wie Abgaben, Studenten, Classrooms anzeigen
	(A21) Der Admin-Bereich muss Konfigurationsseiten für die Schnittstellen zwischen den einzelnen Teilsysteme besitzen
	(A22) Der Admin-Bereich muss Seiten zur Parametrisierung von Tests und Feedback besitzen

Anhang B: Vergleich von NLU-Plattform

Die nachfolgende Evaluation orientiert sich an derjenigen von Canonico und De Russis (2018). Die Bewertungskriterien sind im Kapitel 5.2 beschrieben.

Tabelle 3: Vergleich der NLU-Plattformen Dialogflow, wit.ai, IBM Watson und Amazon Lex

	Usability	Spracherkennung	Frontend-Anbindung	Backend-Anbindung	Preis
Dialogflow	Hoch	15 Sprachen, 60 vordefinierte Entitäten, wie Adressen oder Farben, Ermöglicht Default-Intent Automatische Kontexterkennung	14 Chat-Plattformen werden unterstützt	Webhook und SDK	Standard-Version ist gratis (Anzahl Nachrichten pro Minute auf 180 limitiert, zudem wird Chatbot nach Inaktivität gelöscht) und Business-Version kostet 0,002$ pro Nachricht
Wit.ai	Mittel	50 Sprachen, Keine vordefinierten Entitäten und kein Default-Intent	Keine direkte Online-Integration	SDK	Gratis
IBM Watson	Hoch	12 Sprachen, 7 vordefinierte Entitäten Default-Intent Automatische Kontexterkennung	Keine direkte Online-Integration. Integration per Twilio möglich	SDK	Kostenlos bis zu 10'000 Anfragen pro Monat, danach zahlt man pro Aufruf
Amazon Lex	Niedrig	Nur Englisch, 90 vordefinierte Entitäten von Alexa Automatische Kontexterkennung	Integration per Twilio ermöglicht SMS, FB Messanger und Slack	SDK	Im ersten Jahr gratis

Anhang C: Architekturdokumentation

In diesem Abschnitt befinden sich Anhänge zur Architekturdokumentation. Dazu gehören Beschreibung zu den verwendeten Libraries und der Verzeichnisstruktur sowie eine Abbildung des gesamten Datenbankschemas.

Tabelle 4: Für den Helper-Bot-Server verwendete Node-Libraries

Für den Helper-Bot Verwendete Node-Libraries	
Name, Version und Link	Beschreibung
actions-on-google ^2.6.0 https://www.npmjs.com/package/actions-on-google	Wird von der Library `dialogflow` verwendet, um mit den Google Cloud Services zu interagieren.
axios ^0.18.0 https://www.npmjs.com/package/axios	Promise-basierter http-Client, über welchen Anfragen an die Code-API gemacht werden.
bcrypt-nodejs 0.0.3 https://www.npmjs.com/package/bcrypt-nodejs	Wir für die Speicherung der Passwörter verwendet. Mit der `bcrypt-nodejs` können dazu Hash-Werte erzeugt werden.
body-parser 1.18.3 https://www.npmjs.com/package/body-parser	Der Body-Parser war früher Bestandteil der Express-Library und wird jetzt als eigene geführt. Der Body-Parser nimmt die Anfrage als Middleware entgegen, analysiert den Body und macht diesen unter dem Parameter `req.body` für weitere Middleware verfügbar.
cors ^2.8.5 https://www.npmjs.com/package/cors	Mit dieser Library als Middleware wird CORS (Cross-Origin Resource Sharing) erlaubt, welches für AJAX-Anfragen im Chat-Interface benötigt werden.
dialogflow ^0.8.2 https://www.npmjs.com/package/dialogflow	Diejenige Library, welche die Kommunikation zwischen dem eigenen Chat-Interface und Dialogflow ermöglich. Verwendet `actions-on-google`, um mit den Google Services zu kommunizieren.
dialogflow-fulfillment ^0.6.1 https://www.npmjs.com/package/dialogflow-fulfillment	Die Library Dialogflow Fulfillment erlaubt es einen Dialogflow-Chatbot mit der eigenen Geschäftslogik auszurüsten. Immer dann, wenn eine Absicht erkannt wird, für welche Fulfillment aktiviert wurde, wird ein Webhook an den Helper-Bot versendet und dort die entsprechende Reaktion (z.B. Hilfestellung) erzeugt.
ejs ^2.6.1 https://ejs.co/	Einfache Template-Engine, mit der ein HTML-Markup über JavaScript generiert werden kann. Wird für das Template des Admin-Bereich und für das Chat-Interface benötigt.

express ^4.16.4 https://www.npmjs.com/package/express	Serverseitiges Framework für die Node.js-Umgebung. Bringt Funktionen wie Routing und Middleware mit sich. Siehe Kapitel Architekturdokumentation.
express-basic-auth ^1.1.6 https://www.npmjs.com/package/express-basic-auth	Ermöglich Basic-Auth, eine einfache http-Authentifizierung. Wird für die Authentifizierung von Dialogflow-Fulfillments verwendet.
express-session ^1.15.6 https://www.npmjs.com/package/express-session	Session-Middleware, welche sowohl für das Login, wie auch für die Generierung einer Session für die Interaktion mit Dialogflow benötigt wird.
passport ^0.4.0 https://www.npmjs.com/package/passport	Passport ist eine Express kompatible Authentifizierungs-Middleware für Node.js. Passport verwendet das Konzept der `Strategien` zur Authentifizierung von Anfragen. Die `Strategie` des Helper-Bots basiert auf einer Verifizierung der E-Mail und des Passwortes. Hierzu wird die zusätzliche Library `passport-local` benötigt.
passport-local ^1.0.0 https://www.npmjs.com/package/passport-local	Diese Library ermöglich die Authentifizierung über Benutzernamen und Passwort.
req-flash 0.0.3 https://www.npmjs.com/package/req-flash	`req-flash` erlaubt es Variablen an nachfolgende Aufrufe innerhalb der gleichen Session zu senden. Dies wird dazu verwendet, Erfolgs-, respektive Fehlermitteilung beim Login oder auf den Konfigurationsseiten zu generieren.
sequelize ^5.1.0 https://www.npmjs.com/package/sequelize	Sequelize ist ein ORM, welches als Interface zwischen dem Helper-Bot und der Datenbank dabei hilft Objekte der Anwendung in eine relationale Datenbank abzulegen und auszulesen

Tabelle 5: Anwendungsstruktur des Helper-Bots

Anwendungsstruktur des Helper-Bot-Servers	
Verzeichnis / File	Beschreibung
/config	Enthält sämtliche Konfigurationsfiles, wie beispielsweise für die Datenbank-Verbindung oder die Verbindung zu Dialogflow.
/models	Beinhaltet sämtliche in der Anwendung verwendeten Models. Über das ORM Sequelize können entsprechende Objekte aus der Datenbank gelesen oder darin gespeichert werden. Ein solches Objekt ist beispielsweise ein Student oder eine Konversation.
/public	Der Public-Ordner beinhaltet die im Frontend verwendeten Daten. Dazu zählen CSS- und JavaScript-Files sowie Bilder.

/routes	Die eigentliche Logik der Anwendung befindet sich in diesem Verzeichnis. Darin enthalten ist einerseits ein Unterordner für sämtliche Middleware sowie mehrere Ordner für die einzelnen Komponenten des Helper-Bots. Über eine Route-Funktion wird die Anfrage jeweils and die entsprechende Ressource weitergegeben.
/routes/middleware	In diesem Verzeichnis wird die Middleware gespeichert. Dazu zählen hauptsächlich Funktionen zur Authentifizierung von Benutzern oder Webhook-Anfragen.
/routes/fulfillment	Für jedes Fulfillment gibt es in diesem Verzeichnis einen Unterordner und entsprechende Files. Diese werden je nach erkanntem Intent vom Fulfillment-Handler aufgerufen. Im Unterordner `help` ist das Fulfillment zur Hilfestellung.
/routes/chat	In diesem Verzeichnis befindet sich das Chat-Interface sowie ein Konversationshandler, der für das Senden und Empfangen von synchronen und asynchronen Nachrichten zuständig ist.
/routes/admin	Sämtliche Files, Funktionen und Routers für den Admin-Bereich. Die REST-Schnittstellen für die Live-Daten werden im File `admin.js` definiert und diejenigen für die Konfiguration im Unterordner `config`.
/views	Enthält alle EJS-Templates für den Admin-Bereich und das Chat-Interface.
index.js	Ist dafür zuständig die Anwendung zu initialisieren, indem sie eine Datenbank-verbindung aufbaut, per Express alle nötigen Middlewares definiert und anschliessend die Anfrage an die Router-Funktion weiterleitet.
package.json	Umfasst wichtige Informationen der Anwendung und definiert die Dependencies.

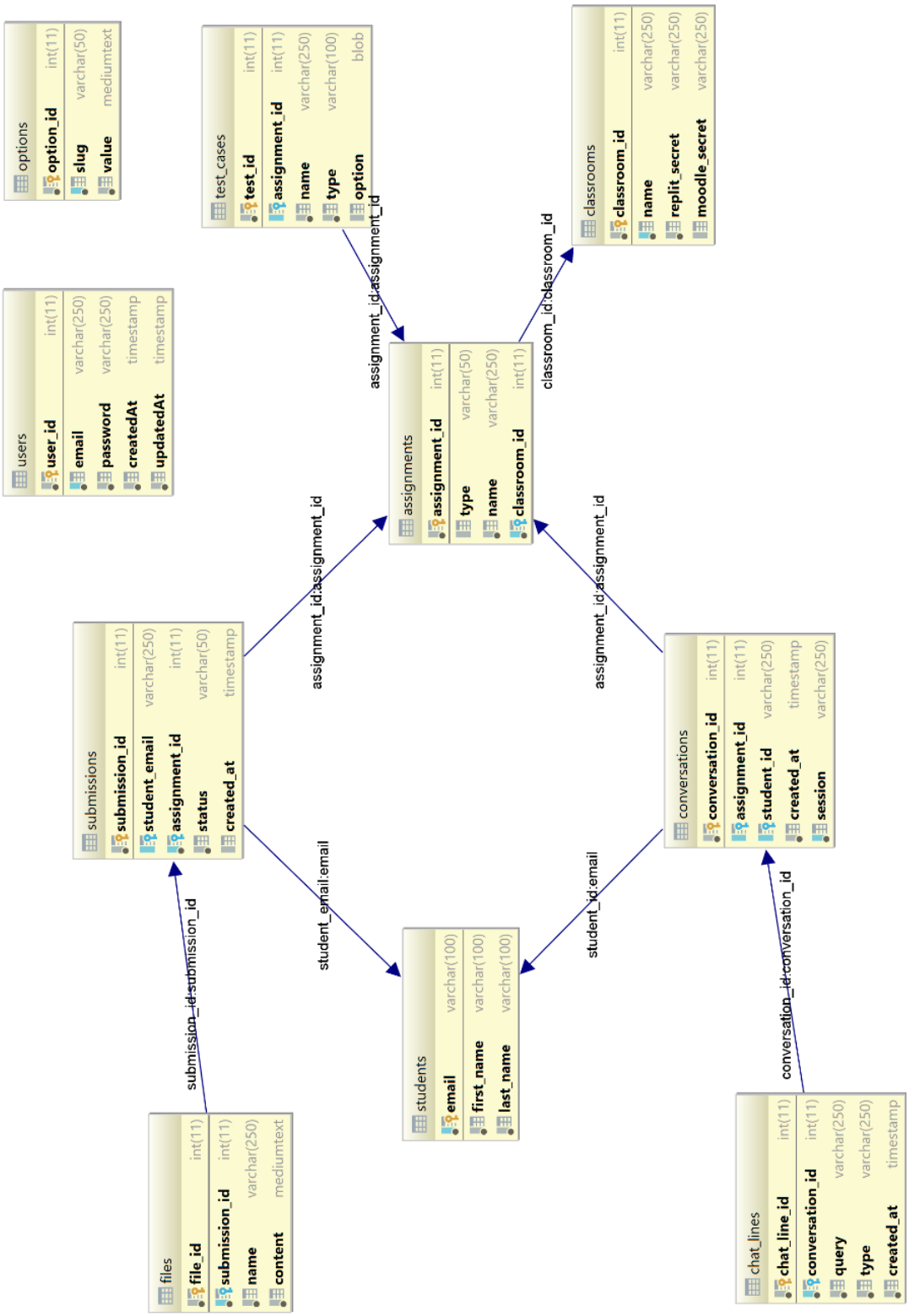


Abbildung 22: Vollständiges Datenbankschema des Helper-Bots

Anhang D: Schnittstellenbeschreibungen

In diesem Anhang sind Schnittstellen des Helper-Bots beschrieben. Zuerst wird jeweils Typ, Format, Endpoint und die Verwendung der Schnittstelle angegeben und anschliessend werden je nach Schnittstelle Request- und Response-Variablen, Codeausschnitte oder sonstige anwendungsbezogenen Informationen aufgeführt.

Tabelle 6: Schnittstellenbeschreibung zwischen repl.it und dem Helper-Bot-Server

<table>
<tr><th colspan="2">Repl.it Webhook</th><th colspan="2">Repl.it → Helper-Bot-Server</th></tr>
<tr><td>Typ</td><td>Webhook per POST (http)</td><td>Format</td><td>JSON</td></tr>
<tr><td>Endpoint</td><td colspan="3">{Helper-Bot-Server}/replit</td></tr>
<tr><td>Verwendung</td><td colspan="3">Der eventbasierte Webhook wird immer dann ausgelöst, wenn ein Student auf repl.it eine Aufgabe zur Überprüfung abgibt (sofern Webhook eingerichtet wurde). Die Daten werden an den Helper-Bot gesendet und dort in die Datenbank geschrieben.</td></tr>
<tr><td>Request</td><td colspan="3">Das JSON-File besitzt folgende Felder:
<table>
<tr><td>Assignment</td><td>Object</td><td>Assignment-Objekt</td></tr>
<tr><td>↳ name</td><td>String</td><td>Name des Assignment</td></tr>
<tr><td>↳ type</td><td>Enum</td><td>Entweder `manual`, `input_output` oder `unit_test`</td></tr>
<tr><td>Classroom</td><td>Object</td><td>Assignment-Objekt</td></tr>
<tr><td>↳ name</td><td>String</td><td>Name des Classrooms</td></tr>
<tr><td>↳ webhook_secret</td><td>String</td><td>Zeichenkette, mit welcher überprüft werden kann, ob es sich um einen unterstützen Webhook handelt.</td></tr>
<tr><td>Submission</td><td>Object</td><td>Submission-Objekt</td></tr>
<tr><td>↳ status</td><td>Enum</td><td>Entweder `submitted`, `submitted_incomplete` oder `complete`</td></tr>
<tr><td>↳ time_submitted</td><td>String (Date)</td><td>Zeitstempel, wann die Submission abgegeben wurde</td></tr>
<tr><td>↳ time_created</td><td>String (Date)</td><td>Zeitstempel, wann die Submission erstellt wurde</td></tr>
<tr><td>↳ teacher_url</td><td>String</td><td>URL des Teacher dieses Assignments</td></tr>
<tr><td>↳ student_url</td><td>String</td><td>URL des Studenten</td></tr>
<tr><td>↳ files[]</td><td>Array</td><td>Array mit sämtlichen abgegeben Files bestehend aus dem jeweiligen Namen und Content.</td></tr>
<tr><td>Student</td><td>Object</td><td>Student-Objekt</td></tr>
<tr><td>↳ first_name</td><td>String</td><td>Vorname des Studenten</td></tr>
<tr><td>↳ last_name</td><td>String</td><td>Nachname des Studenten</td></tr>
<tr><td>↳ email</td><td>String</td><td>E-Mail-Adresse des Studenten</td></tr>
</table></td></tr>
<tr><td>Bemerkung</td><td colspan="3">Leider ermöglicht repl.it zurzeit nur einen einzigen Webhook und zwar dann, wenn der Student eine Aufgabe zur Überprüfung abgibt. Zu diesem Zeitpunkt finden jedoch bereits sämtliche Überprüfungen durch repl.it statt und der Student erhält Bescheid darüber, ob er die Aufgabe richtig gelöst hat (sofern nicht manuelle Bewertung). Für den</td></tr>
</table>

	Studenten kann es dadurch unklar sein, wann und wie ihm der Helper-Bot helfen kann. Falls es zukünftig weitere Webhooks, beispielsweise wenn der Student seinen Code ausführt, geben wird, wäre ein solcher besser für den Helper-Bot geeignet.
Admin-Bereich	Im Admin-Bereich unter `Settings > Classrooms` können diejenigen Classrooms definieret werden, welche vom Helper-Bot unterstützt werden. Hierzu muss der Name und das Webhook-Secret von repl.it kopiert werden. Das Moodle-Secret wird hingegen dazu verwendet, den Aufruf des Chat-Interfaces zu authentifizieren.
Referenzen	https://repl.it/site/docs/classrooms/webhooks

Tabelle 7: Schnittstellenbeschreibung zwischen Code-API und Helper-Bot-Server

Code-API			Code-API ↹ Helper-Bot-Server
Typ	Webhook per POST (http)	**Format**	JSON
Endpoint	{Code-API}/codeAnalysis		
Verwendung	Auf der Code-API wird der Code kompiliert und ausgeführt. Die Code-API wird dazu verwendet den studentischen Code nach syntaktischen und logischen Fehlern zu überprüfen. Die Feedback-Generierung findet auf Seite des Helper-Bot statt.		
Request	Das JSON-File besitzt folgende Felder:		
	Content	String	Zu überprüfender Code
	Name	String	Name des Files
	ID	Int	ID des Files
	Params	Array <String>	Zu überprüfende Input-Parameter
Response	Die Code-API gibt ein JSON mit folgenden Feldern zurück:		
	Content	String	Zu überprüfender Code
	Name	String	Name des Files
	ID	Int	ID des Files
	Params	Array <String>	Überprüfte Input-Parameter
	Problems	Array	Array mit allen Fehlern. Unterscheiden sich je nachdem ob Kompilierungs- oder Laufzeit-Fehler. Kompilierungsfehler beinhalten `position`, `lineNumber`, `source`, `message`, `code`, `startPosition`, `endPosition` und `bug`). Laufzeit-Fehler nur `code`, `message` und `stack_trace`.
	Output	String	Der Output des Programmes. Ist leer, wenn ein Fehler aufgetreten ist.
Bemerkung	Nach Erhalt eines API-Calls wird der Code in ein temporäres File geschrieben, welches anschliessend kompiliert werden kann. Tritt dabei ein Fehler auf, wird dieser zurückgegeben. Ansonsten wird der Code ausgeführt. Zuvor wird der Input- und Output-Stream angepasst. Einerseits soll die übermittelten Werte als Inputs verwendet werden und andererseits, soll das Programm den Output nicht in die Konsole, sondern in eine Variable		

	schreiben. Dieser Output wird anschliessend zurück an den Helper-Bot gesendet, wo dann entsprechendes Feedback generiert wird. Nach der Ausführung wird der Input- und Output-Stream wieder zurückgesetzt und das temporäre File gelöscht. Die Code-API wird für jeden definierten Testfall separat aufgerufen.
Admin-Bereich	Die URL zur Code-API kann über den Admin-Bereich unter `Settings > Java-Code-Analysis` festgelegt werden. Weiter können über diese Seite Hilfestellungen für Java-Fehler definiert werden. Hilfestellungen zur dynamischen Analyse befindet sich hingegen auf der Seite `Settings > Assignments` und dem jeweiligen Assignment.

Tabelle 8: Schnittstellenbeschreibung zwischen Dialogflow und Helper-Bot-Server

Dialogflow Fulfillment			Dialogflow-Fulfillments ↹ Helper-Bot-Server
Typ	Webhook per POST (http)	**Format**	JSON
Endpoint	{Helper-Bot-Server}/fulfillment		
Verwendung	Der eventbasierte Webhook wird immer dann ausgelöst, wenn ein Intent (mit aktiviertem Fulfillment) von Dialogflow erkannt wurde. Anschliessend wird im Helper-Bot anhand des erkannten Intents die jeweilige Ressource aufgerufen. Die Anfrage wird prozessiert und eine entsprechende Rückgabe generiert. In dieser wird angegeben, was Dialogflow als nächstes machen, respektive anzeigen muss. Der oben genannte Endpoint muss in der Dialogflow-Benutzeroberfläche in den Fulfillment-Einstellungen zusammen mit Basic-Auth angegeben werden. Basic-Auth wird auf der nächsten Seite beschrieben.		
Request	Die Anfrage an den Webhook besitzt folgende Felder:		

responseId	String	Eindeutige Request-ID
session	String	Eindeutige Session ID
queryResult	Object	Ergebnis der Sprachverarbeitung
↳ queryText	String	Orginaltext der Anfrage
↳ parameters	Object	Parameter als Name-Value-Paare
↳ allRequiredParamsPresent	Boolean	False, wenn erforderliche Parameter der Abfrage fehlen
↳ fulfillmentText	String	Antworttexte die dem Benutzer angezeigt werden sollen
↳ fulfillmentMessages	Object	Antworten als Rich-Messages z.B. Cards
↳ outputContexts	Object	Sammlung von Ausgabekontexten
↳ intent	Object	Die aus der Benutzereingabe erkannte Absicht.
↳ intentDetectionConfidence	Number 0-1	Genauigkeit des erkannten Intents
↳ diagnosticInfo	Object	Informationen zur Diagnose
↳ languageCode	String	Ausgelöste Sprache
originalDetectIntentRequest	Object	Vollständige Anfrage von einer integrierten Plattform. (Facebook Messenger, Slack, etc.)

<table>
<tr><td>Response</td><td>Die Antwort des Webhook kann folgende Felder besitzen:<table>
<tr><td>fulfillmentText</td><td>String</td><td>Optional. Der Text, der auf dem Bildschirm angezeigt werden soll.</td></tr>
<tr><td>fulfillmentMessages[]</td><td>Object (Message)</td><td>Optional. Die Sammlung von Rich Messages, die dem Benutzer präsentiert werden sollen.</td></tr>
<tr><td>outputContexts[]</td><td>Object (Context)</td><td>Optional. Array mit Ausgabekontexten</td></tr>
<tr><td>followupEventInput</td><td>Object (EventInput)</td><td>Lässt sofort einen weiteren Intent mit dem angegeben Ereignis als Eingabe aufrufen</td></tr>
</table></td></tr>
<tr><td>Node.js Library</td><td>Um Fulfillment zu realisieren, stellt Dialogflow eine Node.js Library zur Verfügung. Diese übernimmt das Generieren einer Response, indem sie diverse Funktionen zur Verfügung stellt. Nachfolgend ist eine beispielhafte Verwendung der Library gezeigt. Weitere Informationen zur Dialogflow Node.js Library kann aus den nachfolgenden Referenzen bezogen werden.<pre>
1. // Import the appropriate class
2. const { WebhookClient } = require('dialogflow-fulfillment');
3.
4. //Create an instance
5. const agent = new WebhookClient({request: request, response: response});
6.
7. agent.add('Herzlich Wilkommen!');
</pre></td></tr>
<tr><td>Basic Auth</td><td>Der Webhook wird zusätzlich mittels Basic-Auth geschützt. Hierzu wird die Node.js Library `express-basic-auth` verwendet. Mit einer Authentifizierungs-Funktion wird geprüft ob, die Benutzerausgabe in der Anfrage mit denjenigen aus der Datenbank übereinstimmen. Die Authentifizierung muss asynchron sein, da der Datenbankaufruf ebenfalls asynchron ist.</td></tr>
<tr><td>Admin-Bereich</td><td>Über den Admin-Bereich können Benutzername und Passwort für die Basic-Auth gesetzt werden.</td></tr>
<tr><td>Referenzen</td><td>https://dialogflow.com/docs/fulfillment
https://github.com/dialogflow/dialogflow-fulfillment-nodejs
https://www.npmjs.com/package/express-basic-auth</td></tr>
</table>

Tabelle 9: Schnittstellenbeschreibung zwischen Chat-Interface und Dialogflow

Chat-Integration			Chat-Interface ⇆ Dialogflow
Typ	gRPC API	**Format**	Node.js Client
Verwendung	Damit Dialogflow mit dem eigenen Chat-Interface interagieren kann, braucht es einen Integrations-Layer. Hierfür stellt Dialogflow den `Dialogflow Node.js Client` zur Verfügung. Diese Clientbibliothek ist gRPC-fähig (Remote Procedure Call, RPC) und unterstützt sowohl REST-, wie auch RPC-Schnittstellen. Die Client-Library benutzt gRPC im Hintergrund für die Kommunikation mit den Google Services und erreicht so einen effizienteren Durchsatz (siehe Referenzen am Ende der Tabelle). Wichtig für die Anwendung ist, dass die Library die Kommunikation mit Dialogflow übernimmt und keine		

	eigenen API-Calls geschrieben werden müssen. Jedoch verlangt es die Interaktion mit den Google-Services, dass ein Dienstkonto eingerichtet wird (siehe Installationsanleitung im Anhang E). Nachfolgend werden die beiden Funktionen zum Senden einer Benutzereingabe und zum Setzen eines Kontexts gezeigt. Beide Funktionen sind Bestandteile der Klasse `Dialogflow` im Ordner `config`.
Benutzereingabe an Dialogflow senden	`/** * Sends text message to Dialogflow */ sendToDialogflow(textMessage, context, lang) { const sessionClient = new dialogflow.SessionsClient({ credentials: { client_email: this.client_email, private_key: this.private_key } }); const sessionPath = sessionClient.sessionPath(this.project_id, this.session_id); const request = { session: sessionPath, queryInput: { text: { text: textMessage, languageCode: lang }, }, queryParameters : { contexts : context, 'resetContexts' : true } }; return sessionClient.detectIntent(request); }`
Bemerkungen zum Senden	Zunächst wird die Verbindung zum Client aufgerufen, indem die E-Mail und der Private-Key übergeben werden. Dann wird über den Projekt-Namen und über eine zuvor generierte Session-ID eine `SessionPath` erstellt, über welchen die Konversation bei weiteren Benutzereingaben identifiziert werden kann.
Kontext setzen	`/** * This function can be used to define the initial context. */ async createContext(context_id, parameters) { contextsClient = new dialogflow.ContextsClient({ credentials: { client_email: this.client_email, private_key: this.private_key } }); const sessionPath = this.sessionClient.sessionPath(this.project_id, this.session_id); const contextPath = contextsClient.contextPath(this.project_id, this.session_id, context_id); const request = { parent : sessionPath, context : { name : contextPath, parameters : struct(parameters), lifespanCount : 5 } }; return await this.contextsClient.createContext(request); }`
Bemerkungen zum Kontext	Für das Setzen eines Kontextes muss neben einem `SessionPath` zusätzlich ein `ContextPath` über den `ContextClient` erstellt werden. Mit `lifespanCount` wird angegeben, über wie viele Benutzerangaben hinweg der Kontext beibehalten wird. Da der Kontext einer bestimmten Struktur folgen muss, werden die Parameter zuvor über die Funktion `struct` aus dem Verzeichnis `config` vorbereitet.

Admin-Bereich	Im Admin-Bereich unter `Settings > Chatbot` müssen folgende Einstellungen gesetzt werden, damit das eigene Chat-Interface mit Dialogflow interagieren kann (siehe Installationsanleitung im Anhang E) Project-ID: Name des Dialogflow-Agenten Service-Account E-Mail: E-Mail-Adresse des Google Dienstkontos Service Account Private Key: Private-Key des Dienstkontos
Referenzen	https://cloud.google.com/dialogflow-enterprise/docs/reference/rpc/ https://www.npmjs.com/package/dialogflow

Anhang E: Installationsanleitung

Dieser Anhang beinhaltet Informationen zur Installation des Helper-Bots verteilet über zwei Tabellen. In einer ersten Tabelle werden die Voraussetzungen erörtert und in einer zweiten die Abfolge von Schritten zur Installation des Helper-Bots gezeigt.

Tabelle 10: Voraussetzungen zur Installation des Helper-Bots

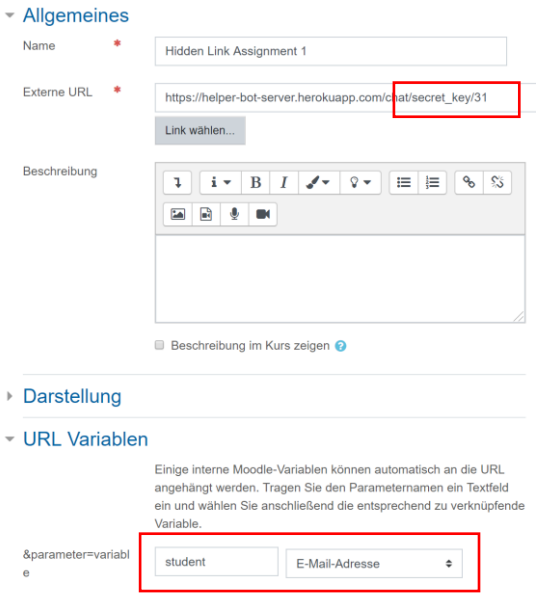
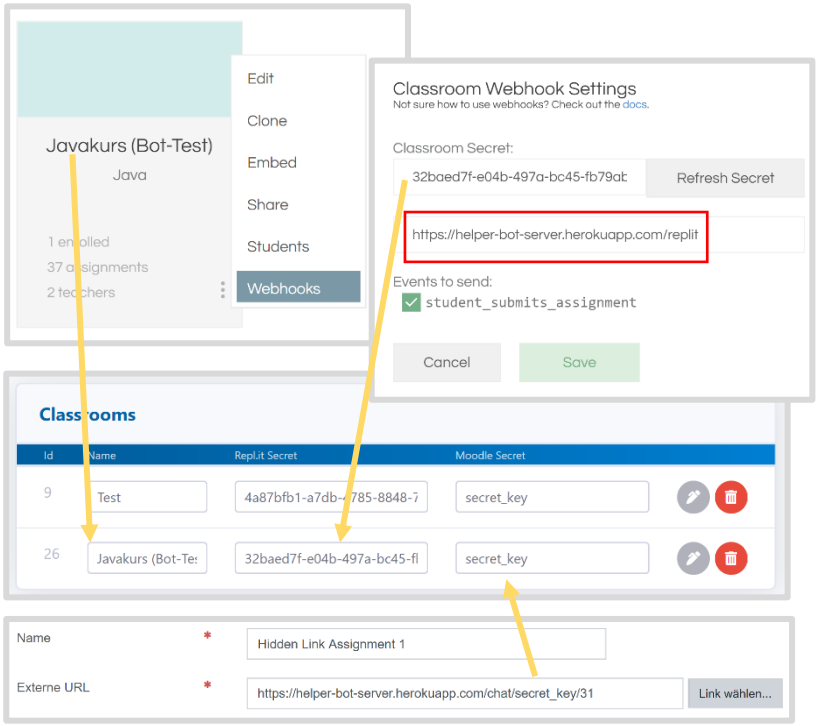
Voraussetzungen für die Installation des Helper-Bots	
Voraussetzung	Beschreibung
Node.js Server	Die eigentliche Helper-Bot Anwendung läuft in einer Node.js-Umgebung. Daher wird ein Server benötigt, der Node unterstützt.
Java Server	Ein zweiter Server wird für die separate Code-API benötigt, welche zur Ausführung des Codes eine Java-Laufzeitumgebung benötigt.
MySQL-Datenbank	Zur Speicherung der studentischen Daten, der Konversationen und der Benutzer des Systems wird eine MySQL-Datenbank benötigt. Für die Einrichtung des Helper-Bots müssen Datenbank-Name, Benutzer, Passwort und Host bekannt sein.
Moodle-Kurs	Die Erstellung von Moodle-Kursen und Kursseiten wird in der Installationsanleitung nicht behandelt und ist vorausgesetzt. Bezüglich der Festlegung von URLs zum Chat-Interface muss folgendes Schema eingehalten werden: {url}/secret_key/31?student=michojan%40students.zhaw.ch Das Secret-Key und die Assignment-Nummer werden jeweils direkt als Teil der URL definiert und die studentische E-Mail über den URL-Parameter `student`. Die entsprechenden Einstellungen eines Moodle-Links sehen wie folgt aus: 

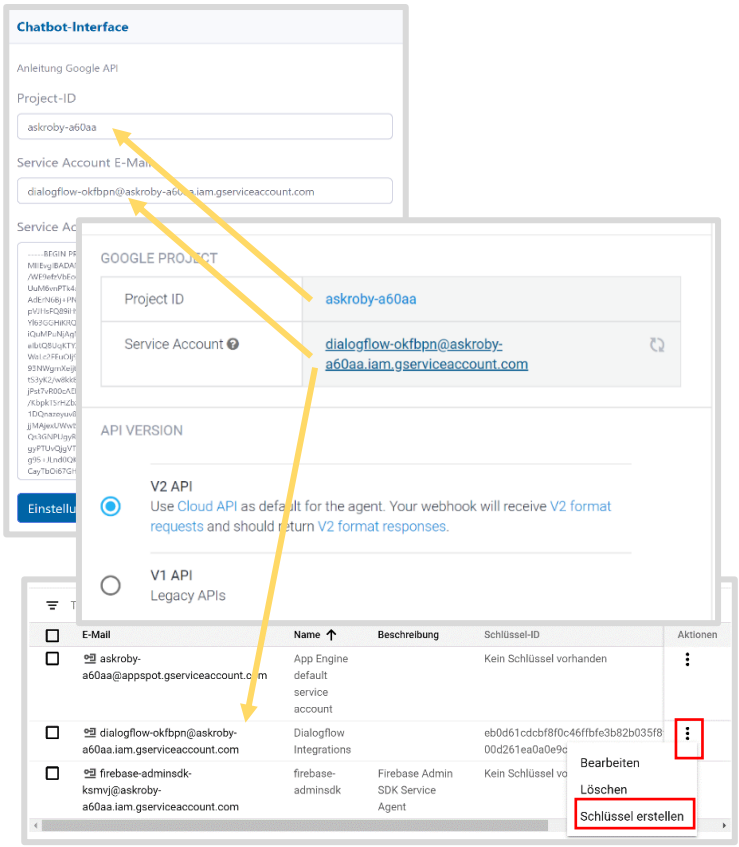
Tabelle 11: Anleitung zur Installation des Helper-Bots

Installationsanleitung des Helper-Bots	
Schritt	Beschreibung
Helper-Bot-Server einrichten	**(1)** Der Code zum Helper-Bot-Server kann über das GitHub-Repository https://github.com/DavidZhaw/helper-bot bezogen werden. Dieser muss anschliessend auf den Node.js-fähigen Server hochgeladen werden.
	(2) Bevor die Anwendung jedoch ausgeführt werden kann, muss zunächst die Datenbank vorbereitet und der Helper-Bot entsprechend konfiguriert werden. Das für den Helper-Bot benötigte Datenbankschema befindet sich im SQL-File database_schema.sql und muss über ein Datenbankverwaltungstool ausgeführt werden. Gleiches gilt für die Standard-Werte im SQL-File initial_data.sql.

	(3) Anschliessend müssen folgende Datenbank-Werte im File `database.js` im Ordner `config` eingetragen werden.
	`// database variables`
	`const database = 'database';`
	`const user = 'user';`
	`const password = 'password';`
	`const host = 'host';`

	(4) Nun sollte über die URL `{host}/admin` das Login-Fenster zum Admin-Bereich zu sehen sein. Über die initialen Daten wurde folgender Benutzer in die Datenbank geschrieben: **E-Mail:** admin@helperbot.com **Passwort:** 1234 Ist das Login mit diesem Benutzer möglich, wurde der Helper-Bot-Server erfolgreich eingerichtet.
Code-API einrichten	**(5)** Der Code für die Code-API ist im Repository https://github.com/zhaw-michojan/helper-bot-code-api verfügbar und muss auf den Java-fähigen Server geladen werden. Es sind keine Änderungen oder Einstellungen an der Applikation notwendig. Wenn die Code-API korrekt installiert wurde, sollte per Aufruf der URL über einen Browser eine Willkommensnachricht angezeigt werden.
	(6) Auf dem Helper-Bot muss unter `Settings > Java Code-Analyse` die entsprechende URL der Code-API eingetragen werden. Diese endet mit `codeAnalysis`
Repl.it Webhook einstellen	**(7)** Für den Austausch zwischen repl.it und dem Helper-Bot, muss auf repl.it der Endpoint des Webhooks und im Admin-Bereich die unterstützen Classrooms angegeben werden. Der Endpoint auf repl.it kann in der Kursübersicht, wie in der nachfolgenden Abbildung zu sehen, festgelegt werden. Anschliessend muss der Name des

	Classrooms und das Classroom-Secret im Admin-Bereich unter `Settings > Classrooms` eigetragen werden.
	(8) Mit dem Moodle-Secret können Aufrufe auf das Chat-Interface reguliert werden. Nur wenn in den Moodle-Einstellungen eines Links das hinterlegte Moodle-Secret enthalten ist, wir der Aufruf autorisiert.
Abbildung: Zusammenspiel repl.it, Helper-Bot und Moodle	
Dialogflow Agent erstellen	**(9)** Als nächstes muss ein Agent auf Dialogflow (https://dialogflow.com/) eingerichtet werden, wozu ein Google-Konto benötigt wird. Für die Erstellung eines neuen Chatbots muss der Name des Bots, die Sprache und die Zeitzone angegeben werden.
	(10) Anschliessend können die Einstellungen des Helper-Bots über die Chatbot-Einstellungen unter dem Punkt `Export und Import` importiert werden. Das zu importierende ZIP-File ist ebenfalls Teil des Helper-Bot-Repository im Verzeichnis `src/dialogflow`. Nachdem der Import bestätigt wurde, werden sämtliche Intents, Entitäten und weitere Einstellungen übernommen.
Dialogflow Chat-Interface einrichten **Refrenzen:** https://medium.com/@tzahi/how-to-setup-dialogflow-v2-authentication-programmatically-with-node-js-b37fa4815d89	**(11)** Sobald der Agent erstellt wurde, muss dieser in den Helper-Bot integriert werden. Im Admin-Bereich unter `Settings > Chatbot` müssen auf der rechten Seite drei Felder ausgefüllt werden. Die einzutragenden Werte erhält man über Dialogflow und den Einstellungen zu einem Chatbot. Dort sollte bereits ein Google-Project mit dem Chatbot verknüpft sein. Falls nicht kann ein neues erstellt oder ein bestehendes verlinkt werden. Wichtig ist anzumerken, dass ein bereits verlinktes Google Project nicht mehr geändert werden kann! Stattdessen muss ein neuer Chatbot erstellt werden. Die drei Felder sind:

	Die einzutragenden Werte der folgenden drei Einstellungen sind in der nächsten Spalte grafisch dargestellt. **Project-ID:** Kann aus den Einstellunten des Chatbots kopiert werden **Service Account E-Mail**: Kann ebenfalls aus den Chatbot-Einstellungen kopiert werden. **Service Account Private-Key:** Per Klick auf die Service-Account-E-Mail gelangt man zur Google Cloud-Plattform und den hinterlegten Dienstkonten (Aufpassen, dass man mit derselben E-Mail, wie zur Erstellung des Chatbots eingeloggt ist). Für die entsprechenden E-Mail muss ein neuer Schlüssel erstellt werden. Der bereits existierende Schlüssel kann nicht wiederverwendet werden! Den Schlüssel als JSON herunterladen. Anschliessend muss mit einem Texteditor der Private-Key aus dem JSON gelesen und in das entsprechende Feld im Admin-Bereich eingetragen werden.
Google Service Account einrichten	
Dialogflow Fulfillments einrichten	**(12)** Damit Dialogflow Fulfillments als Webhook an den Helper-Bot versendet, muss im Dialogflow GUI unter dem Punkt `Fulfillment` die URL des Helper-Bots überschrieben werden: `{host}/fulfillment`
	(13) Weiter kann auf derselben Dialogflow-Seite Basic-Auth für Webhooks eingerichtet werden. Durch den Import wird bereits ein Benutzername eingetragen. Dieser kann angepasst und/oder mit einem Passwort ergänzt werden. Diese beiden Werte müssen im Admin-Bereich unter `Settings > Chatbot` in die Felder auf der linken Seite übertragen werden.

Anhang F: Evaluation des Helper-Bots

Tabelle 12: Anleitung zur Evaluation des Helper-Bots

Abfolge von Schritten zur Evaluation des Helper-Bots		
Nr.	Komponente	Test
(1)	Admin-Bereich	Login im Admin-Bereich möglich? (admin@helperbot.com, 1234)
	Admin-Bereich	Können Java-Fehler unter `Settings > Java Code-Analyse` gesetzt werden?
	Admin-Bereich	Können aufgabenspezifische IO-Tests unter `Settings > Assignments` definiert werden?
(2)	Moodle / Repl.it	Kann eine Programmieraufgabe von repl.it via iFrame auf einer Kurs-Seite bearbeitet werden?
(3)	Helper-Bot-Server / Repl.it	Hat der Webhook von repl.it funktioniert und ist die Lösung nach Submission im Admin-Bereich unter `Assignments` verfügbar?
(4)	Moodle	Verweist der Link auf die richtige Seite und werden alle Parameter übergeben?
	Chat-Interface	Wird das Chat-Interface angezeigt?
	Chat-Interface	Wird der Default-Intent automatisch (nach kurzer Zeitverzögerung) ausgeführt?
	Dialogflow	Wurden die Parameter an Dialogflow übergeben und sind diese entsprechend im Default-Intent vorhanden?
	Chat-Interface / Dialogflow	Reagiert der Chat auf eine Benutzereingabe? Wird zunächst eine Bearbeitungs-Information angezeigt und nach kurzer Verzögerung die Hilfestellung?
(5)	Code-Analyse / Admin-Bereich	Stimmt das tatsächliche Feedback mit dem erwarteten Feedback (entsprechend der Einstellungen im Schritt 1) überein?

Tabelle 13: Untersuchung des Feedbacks mit beispielhaftem Code

Evaluation syntaktischer Fehler

```
1.  import java.util.Scanner;
2.  public class Main {
3.    public static void main(String[] args) {
4.          Scanner keyScan = new Scanner(System.in);
5.          System.out.print("Wort 1: ");
6.          String wort1 = keyScan.nextLine();
7.          System.out.print("Wort 2: ");
8.          String wort2 = keyScan.nextLine();
9.          System.out.print("Wort 3: ");
10.         String wort3 = keyScan.nextLine();
11.
12.         String equalCount = 0;
13.
14.         // TODO: gleiche Worte zählen und in equalCount speichern
15.         if(wort1.equals(wort2) && wort1.equals(wort3) && wort2.equals(wort3)) {
16.           equalCount = 3;
17.         } else if(wort1.equals(wort2) || wort1.equals(wort3) || wort2.equals(wort3)) {
18.           equalCount = 2;
19.         } else
20.           equalCount = 1;
21.         }
22.
23.         System.out.println(equalCount);
24.         keyScan.close();
25.   }
26. }
```

Erwarteter Output:	Zahl kann nicht zu String konvertiert werden
Tatsächlicher Output:	incompatible types: int cannot be converted to java.lang.String
Bemerkung	Fehler wird nach erstmaligem Auftreten in die Fehler-Liste integriert und die Fehlermeldung kann entsprechend ergänzt werden.

Evaluation aufgabenspezifischer Fehler

```
1.  import java.util.Scanner;
2.  public class Main {
3.    public static void main(String[] args) {
4.          Scanner keyScan = new Scanner(System.in);
5.          System.out.print("Wort 1: ");
6.          String wort1 = keyScan.nextLine();
7.          System.out.print("Wort 2: ");
8.          String wort2 = keyScan.nextLine();
9.          System.out.print("Wort 3: ");
10.         String wort3 = keyScan.nextLine();
11.
12.         int equalCount = 0;
13.
14.         // TODO: gleiche Worte zählen und in equalCount speichern
15.         if(wort1.equals(wort2) && wort1.equals(wort3) && wort2.equals(wort3)) {
16.           equalCount = 3;
17.         } else if(wort1.equals(wort2) || wort1.equals(wort3) || wort2.equals(wort3)) {
18.           equalCount = 2;
19.         } else {
20.           equalCount = 1;
21.         }
22.
23.         System.out.println(equalCount);
24.         keyScan.close();
25.   }
26. }
```

Einstellungen Codeanalyse

Name	Input	Output	Ausgabe	Richtig
Alles gleich	Banane	Wort1: Wort2: Wort3: 3	Drei gleiche Worte funktionieren nicht	Ja

	Banane Banane			
Zwei gleiche	Banane Banane Apfel	Wort1: Wort2: Wort3: 2	Zwei gleiche Worte funktionieren nicht	Ja
Keine gleichen	Banane Apfel Birne	Wort1: Wort2: Wort3: 0	Keine gleichen Funktioniert nicht	Ja
Falsch wenn 1	Banane Apfel Birne	Wort1: Wort2: Wort3: 1	Bei drei unterschiedlichen gibt es keine gleichen	Nein

Erwarteter Output:	Keine gleichen Funktioniert nicht Bei drei unterschiedlichen gibt es keine gleichen
Tatsächlicher Output:	Keine gleichen Funktioniert nicht Bei drei unterschiedlichen gibt es keine gleichen

Evaluation aufgabenspezifischer Fehler

```java
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {
        Scanner keyScan = new Scanner(System.in);
        System.out.print("Wort 1: ");
        String wort1 = keyScan.nextLine();
        System.out.print("Wort 2: ");
        String wort2 = keyScan.nextLine();
        System.out.print("Wort 3: ");
        String wort3 = keyScan.nextLine();

        int equalCount = 0;

        // TODO: gleiche Worte zählen und in equalCount speichern
        if(wort1.equals(wort2) && wort1.equals(wort3) && wort2.equals(wort3)) {
          equalCount = 3;
        } else if(wort1.equals(wort2) || wort1.equals(wort3) || wort2.equals(wort3)) {
          equalCount = 2;
        } else {
          equalCount = 0;
        }

        System.out.println(equalCount);
        keyScan.close();
  }
}
```

Einstellungen Codeanalyse

Name	Input	Output	Ausgabe	Richtig
Alles gleich	Banane Banane Banane	Wort1: Wort2: Wort3: 3	Drei gleiche Worte funktionieren nicht	Ja
Zwei gleiche	Banane Banane Apfel	Wort1: Wort2: Wort3: 2	Zwei gleiche Worte funktionieren nicht	Ja
Keine gleichen	Banane Apfel Birne	Wort1: Wort2: Wort3: 0	Keine gleichen Funktioniert nicht	Ja

Falsch wenn 1	Banane Apfel Birne	Wort1: Wort2: Wort3: 1	Bei drei unterschiedlichen gibt es keine gleichen	Nein
Erwarteter Output:	Keine Fehler gefunden			
Tatsächlicher Output:	Keine Fehler gefunden			