HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

IFS INSTITUTE FOR SOFTWARE

ArchBot
architecture is my business

ArchBot

Studien-/Bachelorarbeit HS 2017: Chatbot für Software Design und Review

Abteilung Informatik
Hochschule für Technik Rapperswil

Autoren: Samuel Krieg, Ennio Meier
Betreuer: Prof. Dr. Olaf Zimmermann

Abstract

Ein Chatbot ist ein textbasiertes Benutzerinterface, welches erlaubt, in natürlicher Sprache mit einem System zu kommunizieren. Aktuell werden Chatbots hauptsächlich in den Bereichen Presales und Support verwendet. Die vorliegende Bachelor-/Studienarbeit, sollte evaluieren, ob der Einsatz eines Chatbots im technisch-kreativen Dialog zwischen einem Softwarearchitekten und einem Kunden Unterstützung bieten kann. Dabei sollte der ArchBot Informationen über ein Kundenprojekt sammeln und bewerten, die ein Softwarearchitekt anschliessend für eine an den Chatbot-Benutzer gerichtete, Dienstleistungs-Offerte nutzen kann.

Wir haben eine Vorstudie angefertigt, die einen theoretischen Überblick über die Thematik bietet. Mittels einem Kriterienkatalog verglichen wir in der Vorstudie potenzielle Frameworks zum Erstellen von Chatbots und Natural-Language-Processing-Services zum Erkennen von Benutzerabsichten. Anschliessend testeten wir die zwei geeignetsten Frameworks anhand primitiver Prototypen. Basierend auf den gewonnen Erkenntnissen haben wir ein Framework ausgewählt und anschliessend mit diesem einen Prototypen zu Demonstrationszwecken entwickelt.

Aus der Arbeit hervorgegangen ist ein wiederverwendbarer Kriterienkatalog zur Evaluation von Bot-Development-Frameworks, sowie ein lauffähiger Prototyp, welcher mit dem Benutzer über nicht-funktionale Anforderungen und Architekturmuster diskutieren kann. Ebenfalls wurden Erfahrungswerte dokumentiert, die bei der Umsetzung eines ähnlichen Projekts oder einer Folgearbeit die Entwurfsentscheidungen unterstützen und die Risiken verringern sollen. Die Ergebnisse der Arbeit zeigen, dass sich ein Chatbot nur begrenzt für den untersuchten Anwendungsbereich anbietet: Dialoge zwischen dem Bot und dem Benutzer verlangen strukturierte und vordefinierte Abläufe. Dies erschwert eine kreative und offene Diskussion. Wir sind zum Schluss gekommen, dass für diese Domäne ein Chatbot geeigneter ist, welcher weitere Konzepte aus der künstlichen Intelligenz nutzt und aus einer Vielzahl an Konversationen zwischen Softwarearchitekten und Kunden lernt.

Management Summary

Ausgangslage

Menschen kommunizieren untereinander häufig in einem Frage- und Antwortdialog. Diese Art der Kommunikation fühlt sich für Menschen am natürlichsten an. Chatbots sind grundsätzlich ein Benutzerinterface und versuchen, diese Art der Kommunikation auf Applikationen zu übertragen. Die in den letzten paar Jahren erzielten Fortschritte im Bereich von Natural-Language-Processing (NLP) und Machine-Learning (ML) machen das möglich. Chatbots haben schon in vielen Bereichen Verwendung gefunden. Hauptsächlich werden diese jedoch im Bereich Presales und Support verwendet. Ziel dieser Arbeit ist es, zu evaluieren ob sich Chatbots auch in Bereich Softwarearchitektur behaupten und die Arbeit eines Softwarearchitekten erleichtern können. Es handelt sich bei dieser Arbeit also um eine Machbarkeitsstudie. Ein Softwarearchitekt beschäftigt sich mit der Planung von Softwaresystemen und trifft Entscheidungen über das Zusammenspiel einzelner Komponenten.

Vorgehen und Technologie

Grundsätzlich lässt sich das Projekt in zwei Arbeitsschritte aufteilen. In einem ersten Schritt wurde eine Vorstudie erstellt, in welcher die Theorie von Chatbots behandelt und verschiedene Frameworks evaluiert wurden. Aus dieser Vorstudie resultierten zwei Prototypen, die auf den zwei besten Frameworks basierten. Eine weitere Tätigkeit in diesem Schritt war das Definieren der funktionalen und nicht-funktionalen Anforderungen in Absprache mit dem Betreuer der Arbeit.

Im zweiten Schritt wurde einer dieser beiden Prototypen ausgewählt und weiterentwickelt. Der Bot basiert auf der JavaScript Implementation von Microsoft Bot Framework und greift zur Erkennung von Benutzerabsichten auf Microsofts NLP-Service LUIS (Language Understanding Intelligent Service) zurück. Zusätzlich wurde in diesem Schritt die Architektur spezifiziert und der Bot um eine Datenbankanbindung und Administratortools zum Auswerten der Konversationen bereichert. Auch wurde der vom Betreuer gelieferte Q&A-Content integriert und das Erkennen von Benutzerabsichten mittels NLP auf alle Dialoge ausgeweitet.

Ergebnisse

Das Ergebnis dieser Arbeit lässt sich in verschiedene Teilergebnisse aufspalten. Einerseits wurde eine Vorstudie entwickelt, in welcher Chatbots unabhängig vom Einsatzbereich erforscht werden. Andererseits ist auch ein Bot spezifisch für Softwarearchitektur entstanden. Durch Umsetzen des Bots sind wir auf Einschränkungen gestossen, welche wir bei der Vorstudie nicht in Betracht gezogen haben. So haben wir herausgefunden, dass komplexe Probleme mit geschlossenen Fragen nicht genügend gut besprochen werden können. Unser Bot dient deshalb eher der Informationssammlung und bietet dem Benutzer Anreize zur Selbstreflexion, als dass der Bot effektiv die Lösungsansätze

des Benutzers einschätzt und darauf eingeht. Der Bot kann dabei helfen, nicht-funktionale Anforderungen zu spezifizieren und die Wahl eines Architekturmusters zu besprechen.

Aufgabenstellung

Aufgabenstellung Studien- und Bachelorarbeit (SA, BA)
Samuel Krieg (BA), Ennio Meier (SA)

Chatbot für Software Design und Review («ArchBot»)

1. Auftraggeber und Betreuer

Diese kombinierte Studien- und Bachelorarbeit wird in Zusammenarbeit mit dem Institut für Software (IFS) durchgeführt.

Betreuer HSR:

Prof. Dr. Olaf Zimmermann, HSR FHO, Partner Institut für Software, ozimmerm@hsr.ch

2. Ausgangslage

Bisher werden Chatbots vorranging in Bereichen wie Marketing, Presales und Support eingesetzt; ihre Eignung zur Unterstützung von technisch-kreativen Tätigkeiten und Dialogen soll in diesem Projekt untersucht werden.

3. Ziele der Arbeit und Liefergegenstände

Ein virtueller Question & Answer (Q&A)-Assistent für agile Software-Architekturarbeit mit zwei Top-Level Capabilities soll konzipiert und prototypisch umgesetzt werden: a) Informationssammlung zu Anforderungen und technischen Randbedingungen für eine Offerte im Forward Engineering und b) Architekturreview-Dialog, der in einer Empfehlung für das Service-Offering SW-Healthcheck mündet (im Kontext von Software Evolution & Maintenance).

Die übergeordneten Projektaktivitäten und Liefergegenstände sind:

1. Recherche, Vergleich und Auswahl Chatbot-Framework. Kriterien: u.a. Webfähigkeit, Mächtigkeit der Natural Language Processing (NLP) bzw. Artificial Intelligence (AI) Engine(s), Importmöglichkeiten Content (Formate, APIs), Cloud-Affinität, Zukunftssicherheit, Lizenz, Natural Language Support (Bsp. Englisch/Deutsch).

2. Integration von vorhandenem Q&A Content (wird von Betreuer gestellt, für erste Beispiele siehe https://www.ifs.hsr.ch/index.php?id=13202):

- Business Value und Context
- User Stories
- NFRs/Quality Attribute (siehe z.B. Vorlesung Application Architecture)
- Patterns, Decisions (siehe z.B. Vorlesung Application Architecture)
- Software-Healthcheck

3. Design und Implementierung eines Web-Prototyps auf Basis des ausgewählten Chatbot-Frameworks

Die kritischen Erfolgsfaktoren für diese Arbeit wurden wie folgt definiert:

- agiles Vorgehen, z.B. Umgang mit fortlaufendem Benutzerinput und Feedback
- kognitive Fähigkeiten der erstellten Regeln (Bsp. Eingehen auf Antworten, Qualität Ergebnis/Erfolgsrate)
- AI/Search Metriken wie Precision und Recall (bezogen auf Q&A-Dialogelemente)
- Erweiterbarkeit der Implementierung
- Einholen von und Umgang mit Benutzerfeedback (externer Input/externes Feedback erwartet z.B. von einem Schweizer IT-Dienstleister).

4. Zur Durchführung

Mit dem HSR-Betreuer finden in der Regel wöchentlich Besprechungen statt (Treffen an der HSR oder Telefon- bzw. Webkonferenz). Zusätzliche Besprechungen sind nach Bedarf zu veranlassen.

Alle Besprechungen, bei denen eine Vorbereitung durch den Betreuer nötig ist, sind von den Studenten mit einer Traktandenliste vorzubereiten. Beschlüsse sind in einem Protokoll zu dokumentieren.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. Arbeitszeiten sind zu dokumentieren.

Die Spezifikation der Anforderungen geschieht durch die Studenten in Absprache mit dem Betreuer. Bei Disputen entscheidet der Betreuer in Rücksprache mit den Studenten über die definitiv für die Studien- und Bachelorarbeit relevanten Anforderungen.

Vorstudie, Anforderungsdokumentation und Architekturdokumentation sollten im Laufe des Projektes mittels Milestone mit dem Auftraggeber und dem Betreuer in einem stabilen Zustand abgenommen werden. Zu den abgegebenen Arbeitsresultaten wird ein vorläufiges Feedback abgegeben. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Dokumentation.

Die Rechte an den Ergebnissen der Studien- und Bachelorarbeit werden in einer separaten Vereinbarung definiert (keine Einschränkungen).

Aufgrund des Setups als kombinierte Studien- und Bachelorarbeit ist auf eine arbeitstyp- und aufwandsgerechte Arbeitsverteilung zu achten; diese ist auszuweisen im Projektplan. Alle Liefergegenstände und Zwischenergebnisse (Bsp. Software-Komponente, Berichtskapitel) sollten einen eindeutig identifizierbaren Verantwortlichen haben, dem der Projektpartner dann zuarbeiten kann. Ob ein oder zwei technische Berichte abzugeben sind, wird im Projektverlauf festgelegt.

5. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Bei der Projektdokumentation und Ihrer Abgabe sind die „Allgemeine Informationen

zu Studien- und Bachelorarbeiten" sowie die „Anleitung: Dokumentation Studien- und Bachelorarbeite" inklusive Anhängen (jeweils auf dem HSR Intranet verfügbar) zu beachten.

6. Termine (Quelle: HSR Intranet)

18.09.2017	Beginn der Studien- und Bachelorarbeit, Ausgabe der Aufgabenstellung durch die Betreuer
	Fotoshooting. Genauere Angaben erteilt die Kommunikationsstelle rechtzeitig.
19.12.2017	Abgabe Kurzbeschreibung (Abstract für Diplomarbeitsbroschüre) an das Abteilungssekretariat. Die Studierenden senden per Email das A0-Poster zur Prüfung an ihren Examinator/Betreuer.
22.12.2017	Abgabe des Berichtes an den Betreuer bis 12.00 Uhr. Fertigstellung des A0-Posters bis 12.00 Uhr. Abgabe des Posters im Studiengangsekretariat.
08.01 – 02.02.18	Mündliche BA-Prüfung (nur S. Krieg), voraussichtlich in der Beratungswoche

Allfällige weitere Termine sind am Sekretariat der Abteilung Informatik zu erfragen und sollten entsprechend in einem Sitzungsprotokoll dokumentiert werden.

7. Beurteilung

Eine erfolgreiche Studienarbeit zählt 8 ECTS-Punkte pro Studierenden. Eine erfolgreiche Bachelorarbeit zählt 12 ECTS-Punkte pro Studierenden. Für 1 ECTS-Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert. Siehe auch Modulbeschreibung der Studien- und Bachelorarbeiten.

Für die Beurteilung ist der HSR-Betreuer verantwortlich. Aufgrund des Setups als kombinierte Studien- und Bachelorarbeit werden mglw. zwei Einzelnoten vergeben.

Im Übrigen gelten die Bestimmungen der Abt. Informatik zur Durchführung von Studien- und Bachelorarbeiten.

Rapperswil, den 19. 09. 2017

Prof. Dr. Olaf Zimmermann
Institut für Software
Hochschule für Technik Rapperswil

Inhaltsverzeichnis

Glossar

Built-In-Datenbank	beschreibt eine Datenbank, die bereits in einer Software eingebunden ist..
Carousel	beschreibt eine UI-Komponente, welche mehrere Inhalte darstellt, durch welche mit scrollen navigiert werden kann..
Chatbot	Textbasiertes Benutzerinterface, welches erlaubt mit natürlicher Sprache mit einem System zu kommunizieren.
CRUD	Grundlegende Datenbankoperationen (lesen, ändern, schreiben, löschen).
Docker-Host	Gerät, auf dem ein Docker-Image läuft.
Docker-Image	portable Abbildung eines Containers (virtuelles Betriebssystem, auf dem nur eine Applikation läuft).
Entity	stellt eine Instanz eines Objekts dar..
Intent	die Absichten eines Benutzers, welche von einem NLP-Service aus einer Nachricht gefiltert wird..
JSON	JavaScript Object Notation, kompaktes Datenformat zum Zweck des Datenaustauschs zwischen Anwendungen.
Machine-Learning (ML)	ermöglicht, dass ein System aus Wissen aus vorgegebenen Beispielen generiert und dieses auf neue Beispiele anwenden kann..
Messaging Plattform	Plattformen, vorwiegend zur Kommunikation mit anderen Menschen, wie z.B. WhatsApp, Telegram..
Natural-Language-Processing (NLP)	Fähigkeit eines Systems, die Sprache zu verstehen, wie sie geschrieben bzw. gesprochen wird.
ORM	Objektrelationale Abbildung.
Q&A-Content	Inhalt, welcher vom Betreuer geliefert wird..
RESTful	Methode, um die Kommunikation zwischen einem webbasierten Client und Server zu ermöglichen..
Software-Development-Kit (SDK)	Eine Sammlung von Programmierwerkzeugen und Programmbibliotheken, die zur Entwicklung von Software dienen..

Thread	beschreibt eine Ausführungsreihenfolge eines Ablaufs..
Universal-Message-Markdown	beschriebt eine schlanke Markdown-Language..
Web-Endpoint	Pfad, um von externen Client-Applikationen auf einen Service zuzugreifen..
WordPress	ist ein weit verbreitetes Open-Source Inhaltsverwaltungssystem für Webseiten..

1. Einleitung

In diesem Dokument ist die Bachelorarbeit von Samuel Krieg und die Studienarbeit von Ennio Meier dokumentiert, welche an der Hochschule für Technik Rapperswil (HSR) im Herbstsemester 2017 absolviert wurde. Die Arbeit wurde von Prof. Dr. Olaf Zimmermann (Professor, Institutspartner des Instituts für Software der Hochschule für Technik Rapperswil) betreut.

1.1. Einführung

Mit der steigenden Digitalisierung der Gesellschaft verlagert sich die Kommunikation vermehrt auf digitale Kommunikationskanäle. Diese Transformation bringt neue Herausforderungen und Möglichkeiten, die es zu bewältigen und nutzen gilt. Diese Veränderung hat grosse Auswirkungen auf die Gesellschaft.

Vor einer Generation, als die meisten Menschen nur ein Festnetzanschluss besassen, konnte ein verpasster Telefonanruf unproblematisch ein paar Stunden später beantwortet werden. Durch die starke Verbreitung von Smartphones und der dadurch gestiegenen Erreichbarkeit, hat sich dieser Zeitrahmen verkleinert. Die Gesellschaft hat ihre Vorstellung von Kommunikation im Bezug auf Erreichbarkeit und Geschwindigkeit drastisch geändert. Eine der Folge davon ist, dass auch die Erwartungshaltung an Dienstleistungen diesen Kriterien gerecht werden müssen.

Ebenfalls ermöglicht der Fortschritt der Digitalisierung neue Bereiche zu automatisieren, welche bis anhin nur von einer menschlichen Arbeitskraft abgedeckt werden konnten. Somit bietet die Digitalisierung der Wirtschaft eine Möglichkeit, dem Drang nach Kostenoptimierung nachzugeben.

An diesem Punkt kommen Chatbots ins Spiel. Sie können die Erwartungshaltung an immer erreichbare und schnelle Dienstleistungen befriedigen und reduzieren gleichzeitig die Kosten für den Chatbot-Anbieter. Zusätzlich kann mit Chatbots eine persönliche Beratung des Kunden erzielt werden.

1.2. Vision

Die Softwarearchitekturplanung und Review bereits bestehender Architektur wird oft in einem technisch-kreativen Dialog zwischen Kunden und Softwarearchitekten durchgeführt. Diese Gespräche erfordern vom Kunden eine Vorbereitung und besitzen aus der Sicht des Softwarearchitekten immer eine ähnliche Grundstruktur. Ein Chatbot kann Teile des Dialogs mit dem Kunden übernehmen und dadurch den Prozess durch Vermitteln von Wissen und Sammeln von Informationen beschleunigen. Ebenso bietet sich so für den Softwarearchitekten die Möglichkeit, bereits mit Hintergrundinformationen zum Projekt in das Gespräch einzusteigen.

1.3. Was sind Chatbots?

Ein Chatbot ist grundsätzlich eine Art von User Interface, das mit einem Service interagiert, wie auch eine Webseite eine Art von Interface ist. Der Kommunikationsaustausch findet text- oder immer häufiger auch sprachbasiert statt. Diese Art der Kommunikation fühlt sich für die Benutzer am natürlichsten an, da die zwischenmenschliche Kommunikation ebenfalls in einem Frage-/Antwort-Dialog stattfindet.

Einer der Hauptgründe für das verstärkte Auftreten von Chatbots ist, dass sich das Benutzerverhalten geändert hat. Benutzer greifen immer häufiger über Smartphones auf Services zu. Gleichzeitig benutzen die Benutzer immer weniger Apps, was das Einbinden von Services in verbreitete Apps sinnvoll macht. Parallel zu dieser Veränderung des Benutzerverhalten hat sich die Technologie im Bereich von Natural-Language-Processing und Machine-Learning stark weiterentwickelt und ermöglicht somit das Umsetzen von intelligenten Chatbots mit einer hohen Erfolgsquote für das Erfüllen der Benutzeranfragen[16].

1.4. Struktur des Dokuments

Das Dokument ist neben diesem Kapitel in fünf weitere Hauptkapitel unterteilt. Das Kapitel «Anforderungen» beinhaltet die funktionalen und nicht-funktionalen Anforderungen, die unser Chatbot erfüllen soll. Das Kapitel «Vorstudie» macht einen grossen Teil unserer Arbeit aus. Die Vorstudie kann als Anleitung für das Umsetzen von guten Chatbots angesehen werden. In diesem Kapitel widmen wir uns zusätzlich den verschiedenen Bot-Development-Frameworks und den zwei darauf aufbauenden Prototypen. Die «Architekturdokumentation» ist das nächste Kapitel und dokumentiert sowohl die logische Architektur, wie auch das Deplyoment des ArchBots. Im darauffolgenden Kapitel «Umsetzung» dokumentieren wir, wie wir den ArchBot implementiert haben. Im letzten Kapitel «Reflexion» gehen wir auf die Ergebnisse und Erkenntnisse dieser Arbeit ein. Ausserdem wagen wir in diesem Kapitel einen kritischen Rückblick auf unsere Arbeit und einen Ausblick in die Zukunft, wie man den ArchBot erweitern könnte.

2. Anforderungen an den ArchBot

In der Anforderungsanalyse haben wir entscheidende funktionale und nicht-funktionale Anforderungen für das Projekt definiert.

2.1. Funktionale Anforderungen

Wir haben die funktionalen Anforderungen als User-Stories beschrieben.

2.1.1. Informationssammlung

Als Softwarearchitekt möchte ich vom Chatbot durch ein Gespräch geführt werden, in dem Softwaredesignaspekte beleuchtet und Informationen über ein geplantes oder bestehendes Projekt gesammelt werden, so dass ein Administrator aus den von mir gewonnenen Antworten eine Offerte für mich erstellen kann.

2.1.2. Auslesen der Konversationen

Als Administrator möchte ich die Konversationen der Softwarearchitekten auslesen können, in dem jede Konversation abgespeichert wird, um die Informationen für das Erstellen einer Offerte nutzen zu können. Ich möchte alle Antworten auf einzelne Fragen sehen können, um Rückschlüsse auf die Qualität der Frage stellen zu können.

2.2. Nicht-Funktionale Anforderungen

2.2.1. Performance

Antwortzeit Die Antwortzeit des Chatbots soll unter einer Sekunde sein, bei längeren Wartezeiten durch komplexe Operationen (z.B. Abfrage einer Datenbank) Rückmeldung geben, dass er arbeitet (z.B. «Ich suche in meiner Datenbank und melde mich gleich»). Die maximale Antwortzeit, in welcher der Chatbot die gewünschte Antwort liefert, beträgt zehn Sekunden. Die durchschnittliche Antwortzeit beträgt drei Sekunden.

2.2.2. Compatibility

Laufzeitumgebung Der Chatbot muss auf Windows lauffähig sein.

Channel Integration Einbinden des Chatbots in bestehende Channels wie z.B. Slack, FB Messenger, Telegram Messanger innerhalb von einem halben Tag ohne Änderungen im Code umsetzbar.

2.2.3. Usability

Feedback Der Chatbot soll mit unerwarteten/unklaren Inputs umgehen können und als Antwort ein Beispiel von möglichen Anfragen machen.

Ease of Use Ein neuer Benutzer benötigt in den ersten fünf Minuten maximal zweimal Hilfestellung. Ein neuer Benutzer macht in den ersten fünf Minuten maximal drei Fehler beim Benutzen des Chatbots.

Sprache Die Kommunikation mit dem Chatbot findet in Englisch statt. Die Sprache des Chatbots ist angemessen für die Zielgruppe (Softwarearchitekten).

Kommunikationsart Die Kommunikation mit dem Chatbot findet schriftlich statt.

Satisfaction Chatbot begrüsst Benutzer und fragt nach dem Namen, Chatbot spricht danach den Benutzer persönlich an.

2.2.4. Transferability

Installation Chatbot kann mit der Installationsanleitung innerhalb von zehn Minuten installiert werden.

2.2.5. Accessability

Erkennung der Absicht des Benutzers Chatbot erkennt in 80% der Fälle die Absicht des Benutzers.

2.2.6. Maintainability

Zukunftssicherheit Langfristig verfügbare Komponenten, welche schon seit mindestens zwei Jahren bestehen und von ein aktiven Entwicklungsteams getragen werden.

3. Vorstudie zum Thema Chatbot

In der Vorstudie behandeln wir die grundlegende Theorie von Chatbots, welche Faktoren für das Entwickeln eines Chatbots bedeutend sind, welche Arten von Chatbots es gibt und welche Bot-Development-Frameworks für das Projekt in Betracht zu ziehen sind. Auch haben wir Tests definiert, um Frameworks auf Erfüllen dieser Faktoren zu testen. Neben einem Kriterienkatalog zur Evaluation von technischen Faktoren der Frameworks beschreiben wir in der Vorstudie ein Prototypevaluationsszenario, welches später dabei hilft, die Prototypen für unseren spezifischen Use-Case zu testen. Ein weiterer Teil der Vorstudie ist das Definieren von Testmetriken, welche vom Chatbot erfüllt werden müssen. Das Projekt war stark von der Vorstudie abhängig und deshalb war es wichtig, bei der Vorstudie sorgfältig vorzugehen.

3.1. Einführung in Thematik von Chatbots

In diesem Unterkapitel beschreiben wir, welche Arten von Chatbots es gibt, was ein Bot-Development-Framework ist, wie die Architektur eines Chatbots umgesetzt werden könnte und wie Konversationen aufgebaut sein sollten.

3.1.1. Arten von Chatbots

Grundsätzlich kann man Chatbots in verschiedene Arten unterteilen[9, 17, 5].

Retrieval-Based Chatbot ist trainiert auf einen Satz möglicher Fragen und möglicher Antworten. Der Chatbot findet für jede Frage die am relevanteste Antwort aus allen möglichen Antworten. Der Chatbot kann jedoch nicht selbstständig neue Antworten generieren. Die Qualität nimmt mit Anzahl Fragen zu. Antworten sind vorgeschrieben und deshalb sprachunabhängig. Ausserdem muss der Chatbot sich nicht um Rechtschreibung und Grammatik kümmern.

Generative-Based Können im Gegensatz zum retrieval-based Model auch selbstständig Antworten generieren. Das macht den Chatbot intelligenter, weil Wort für Wort des Inputs verarbeitet wird. Das zieht jedoch den Nachteil mit sich, dass die Rechtschreibung und die Grammatik fehlerhaft sein kann. Wenn der Chatbot präzise trainiert wurde, ist dieser besser als das retrieval-based Modell, weil er mit komplexen Fragen und unerwarteten Inputs umgehen kann.

Open/Closed Domain Zudem können die Konversationen der Chatbots in open- und closed-domain aufgeteilt werden. Im Gegensatz zu open-domain Chatbots, welche Fragen zu jeglichen Themen beantworten können, kann ein closed-domain Chatbot nur zu bestimmten Themen Fragen beantworten.

Für unser Projekt ergibt ein retrival-based closed-domain Modell am meisten Sinn, da wir vordefinierte Fragen und dazu passende Antwortmöglichkeiten haben und uns auf ein bestimmtes Thema festlegen (Softwarearchitektur).

3.1.2. Bot-Development-Frameworks

Um einen Chatbot zu entwickeln, braucht es zahlreiche Komponenten. Dazu gehört neben einer Web-App für den Chatbot auch eine Schnittstelle zu Messaging Plattforms und Natural-Language-Processing, um die Absicht (Intent) des Benutzers aus der Nachricht herauszufiltern[9]. Ein Bot-Development-Framework erleichtert und beschleunigt den Entwicklungsprozess, indem es diese Komponenten bereitstellt.

3.1.3. Grundlegende Chatbot Architektur

Der Aufbau, sowie der Ablauf von vielen Chatbots ist sehr ähnlich, unterscheidet sich dennoch von Chatbot zu Chatbot und ist abhängig von der Umgebung, beziehungsweise der eingesetzten Technologie. Wir haben die Beschreibung der Komponenten und des Ablaufs bewusst möglichst generisch gehalten.

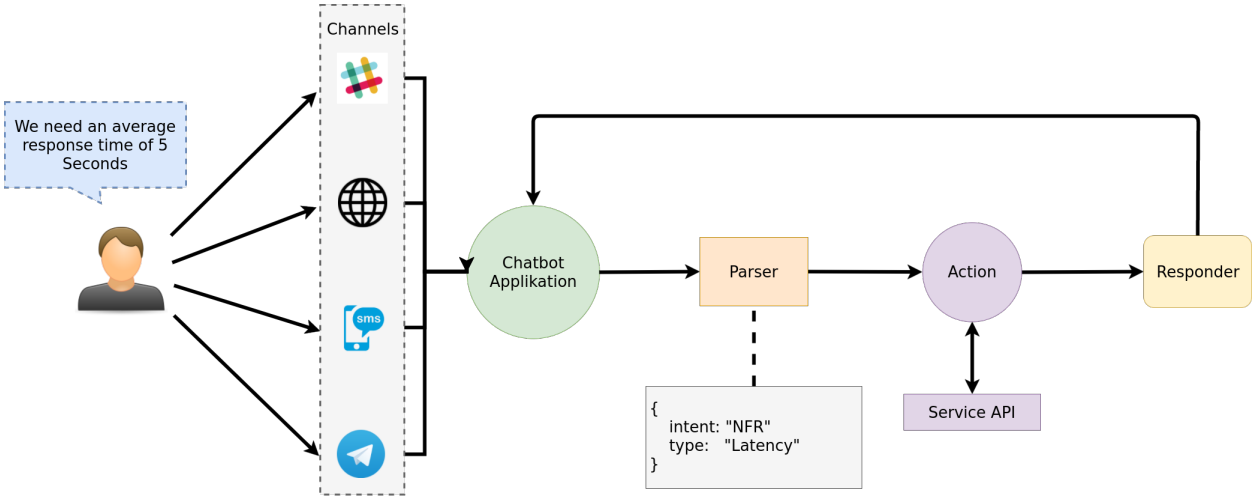


Abbildung 3.1.: Darstellung einer generische Chatbot Architektur adaptiert von Beitrag auf Quora[14]

Wie aus der Abbildung 3.1 ersichtlich ist, gibt der Benutzer eine Nachricht über einen der Channels ein. Die Nachricht wird von der Chatbot-Applikation an den Parser weitergeleitet, welcher die Absicht des Benutzers erkennt. In diesem Fall will der Benutzer eine nicht-funktionale Anforderung erfassen. Die Absicht des Benutzers wird an die Action weitergeleitet, die entscheidet, was mit der aus dem Parser gewonnenen Information geschieht. Anschliessend wird vom Responder eine für den Benutzer leserliche Antwort generiert und über die Chatbot-Applikation zurück an den Benutzer gesendet.

Benutzer Einer der wichtigsten Bestandteile eines Chatbots ist der Benutzer. Er interagiert mit dem Chatbot und hat Erwartungen, die erfüllt werden müssen.

Channels Die Channels dienen der Verbindung des Benutzers zum Chatbot. Als Medium zur Kommunikation mit dem Chatbot wird natürliche Sprache sowohl in geschriebener, wie auch in gesprochener Form verwendet. Channels mit geschriebener Sprache als Medium sind zum Beispiel Chat-Platformen wie Facebook, Slack und Telegram. Bekannte Channels, die mit Sprache in gesprochener Form arbeiten sind: Cortana, Alex, Siri. Ein Chatbot kann mehrere dieser Channels bedienen und somit seine Reichweite vergrössern. Channels werden manchmal auch als Interfaces bezeichnet[8].

Chatbot-Applikation Die Chatbot-Applikation beginnt eine Konversation über den vom Benutzer gewählten Channel. Ebenfalls kümmert sich die Applikation um das Session-Handling und reicht Nachrichten, welche von den Channels stammen, an den Parser weiter.

Parser Die Nachrichten müssen von einem dem Menschen verständlichen Format in strukturierte Daten umgewandelt werden, welche anschliessend zur Weiterverarbeitung bereitstehen. Diese Umwandlung findet mit NLP statt[4]. Dabei werden aus den Nachrichten meist zwei Informationen herausgefiltert. Als erstes wird der Intent der Nachricht ausgelesen. Intents dienen dem System dazu festzuhalten, was der Benutzer beabsichtigt und welche zugehörige Action das System auszuführen hat[11]. Ist die Absicht des Benutzers klar, werden die, wenn vorhanden, zum Intent gehörigen «Entities» aus der Nachricht extrahiert. Eine Entity stellt ein Objekt in der Aussage dar[7]. Als Beispiel geben wir dem Parser den folgenden Satz als Input: «Finde alle asiatischen Restaurants im Umkreis von 10 km.» Als Output des Parsers werden wir folgendes erhalten:

- Intent: Restaurant finden
- Entities:
  - Küche: Asiatisch
  - Entfernung: 10km

Action Nach dem Parsen werden Actions von Intents ausgelöst. Eine Action definiert, was an einer Service-API oder sonstige Software für eine Aktion ausführt werden soll. Die Entities werden als Parameter der Action mitgegeben und können von der Software verarbeitet werden[2].

Responder Der Responder bekommt Daten von einer Action und muss diese, in ein für den Benutzer verständliches Format, übersetzen. Konkret bedeutet das, dass der Responder die Antwort auf die Eingangsmessage des Benutzers formuliert. Ist die Antwort formuliert, wird diese an die Applikation gesendet, welche sie an den Benutzer weiterleitet.

3.1.4. Konversation

Die Verwendung von Sprache ist für den Menschen ein natürliches Mittel zur Interaktion. Diese Interaktion setzt jedoch das Einhalten bestimmter Normen und Regeln voraus, um eine reibungslose Kommunikation zu ermöglichen.

Bestandteile einer Konversation

1. **Onboarding** In dieser Gesprächsphase stellt sich der Chatbot vor und erklärt, was sein Zweck ist und wie man mit ihm interagieren kann. Es muss für den Benutzer klar erkennbar sein, dass er mit einem Chatbot kommuniziert und nicht mit einen Menschen. Der Chatbot stellt in dieser Phase Fragen über den Benutzer, um den Gesprächsverlauf personalisieren zu können[19].

2. **Funktionalität** In dieser Phase findet der eigentliche Frage-/Antwortdialog statt. Für jede Funktionalität (z.B. Sammeln der nicht-funktionalen Anforderungen) wird der Dialogfluss festgelegt.

3. **Feedback und Error Handling** Der Chatbot muss über ein sinnvolles Errorhandling verfügen und auch Feedback zu den Benutzereingaben geben können[1].

4. **Unterstützung** Der Chatbot sollte den Benutzer auf Probleme hinleiten und der Benutzer muss zu jedem Zeitpunkt der Konversation Unterstützung vom Chatbot anfordern können[15].

Konversationsfluss

Das Design eines Konversationsflusses ist mitunter einer der schwierigsten Punkte beim Erstellen eines Chatbots. Ziel ist es, durch möglichst wenig und kurzen Nachrichten mit dem Benutzer, die Erwartungen an den Chatbot zu erfüllen. Dies ist insbesondere eine Herausforderung, weil die Konversation viele verschiedene Wege nehmen kann, um ein bestimmtes Ziel zu erreichen. Um den Konversationsfluss festzuhalten, bietet sich die Verwendung eines Flussdiagramms an. Als Inspiration diente der Blogbeitrag von Jesús Martín[10].

Wie aus der Abbildung 3.2 ersichtlich ist, gliedert sich eine Konversation in verschiedene Dialoge, welche durch die verschiedenen Benutzerabsichten (Intent) ausgelöst werden. Zu jedem Dialog kann idealerweise eine Hilfestellung angefordert werden. Sobald ein Dialog gestartet ist, läuft ein einfacher Frage-/Antwortdialog zwischen dem Benutzer und dem Chatbot ab oder es werden alternative Interaktionsmöglichkeiten angeboten (z.B. ein Carousel mit einer Auswahlliste, Button). Sobald ein Dialog abgelaufen ist, wartet der Chatbot wieder darauf, neue Benutzerabsichten zu erkennen und den passenden Dialog zu starten.

Es gilt verschiedene Gesichtspunkte zu beachten, um einen reibungslosen Konversationsfluss zu ermöglichen:

Prägnante Formulierung Der Chatbot sollte sich immer möglichst prägnant ausdrücken. Muss der Benutzer nachfragen, weil er etwas nicht verstanden hat oder muss er zu viel lesen, verschlechtert dies die User-Experience beträchtlich[15].

Hilfestellung anbieten Zu jedem Moment kann der Benutzer Hilfe vom Chatbot anfordern. Beispielsweise tritt dies ein, wenn der Benutzer nicht mehr weiss, wo er sich in der Konversation befindet oder was seine Möglichkeiten sind. Der Chatbot soll in diesem Fall den Benutzer unterstützen, beispielsweise durch das Aufzeigen der aktuell möglichen Aktionen des Benutzers. [15]

Fallback-Szenario Kann der Chatbot eine Absicht des Benutzers nicht erkennen, soll ein Fallback-

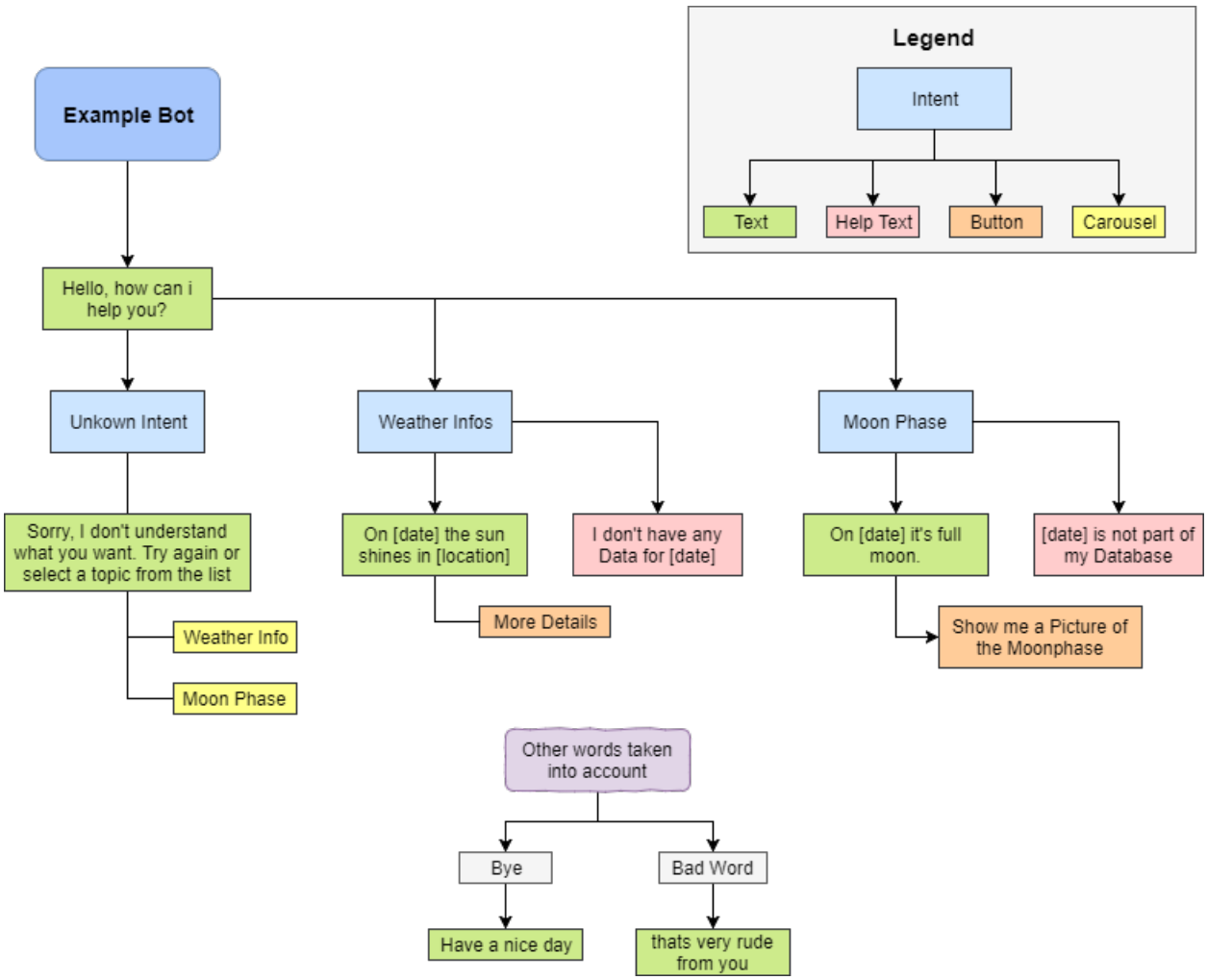


Abbildung 3.2.: Beispiel Konversationsfluss

Szenario definiert sein, welches den Benutzer auf einen gültigen Pfad der Konversation führt. Dies kann beispielsweise durch Aufzählung der aktuellen Möglichkeiten geschehen[3].

Verständnis überprüfen Wenn möglich sollte der Chatbot überprüfen, ob er den Benutzer richtig verstanden hat. Das kann durch Repetieren des Benutzer-Inputs in der Antwort geschehen oder es kann eine explizite Frage an den Benutzer gestellt werden. Beispiele: «Hier ist Ihr heutiger Wetterbericht für Jona: ..., Gehe ich richtig in der Annahme, dass Sie gerne den heutigen Wetterbericht für Jona wissen möchten?». Entsteht ein Missverständnis ist dies, je weiter eine Konversation fortgeschritten ist, umso schwieriger zu korrigieren. [15]

Standpunkt Zu gutem UX-Design klassischer Software gehört es, dem Benutzer stets den Standpunkt im Prozess mitzuteilen. Das trifft auch auf Konversationen mit dem Chatbot zu. Dies kann durch aktive Kommunikation des Chatbots geschehen oder durch eine Anzeige des aktuellen Standpunktes. Ebenfalls sollte der Benutzer jederzeit den Chatbot fragen können, wo er sich befindet. [15]

Alternative Interaktionsmöglichkeiten In gewissen Situationen ergibt es Sinn, dem Benutzer al-

ternative Interaktionsmöglichkeiten zu bieten. Muss der Benutzer sich im Dialog zwischen drei Möglichkeiten entscheiden, kann dem Benutzer zusätzlich zum Chat, die Auswahl als Knöpfe zum Betätigen dargestellt werden[15, 1].

Häufige Fragen Der Chatbot sollte auf Fragen, welche häufig gestellt werden, wie z.B «Wie heisst du?» oder «Wie geht es dir?», antworten können[6].

Typische Gesprächswörter Der Chatbot sollte in der Lage sein, auf typische Gesprächswörter wie «Hallo», «Bye», «Restart» reagieren zu können. [6].

3.2. Kriterienkatalog

Um die Bot-Development-Frameworks evaluieren zu können, definierten wir Kriterien, welche uns für unser Projekt wichtig waren. Mit einer anschliessenden Nutzwertanalyse haben wir die Auswahl an Frameworks mittels dieser Kriterien eingeschränkt.

Folgende Kriterien haben wir in Betracht gezogen:

Mögliche Channels Der Chatbot muss unter Windows lauffähig sein. Der Chatbot sollte idealerweise in möglichst viele Channels (Telegram, Slack, usw.) integriert werden können.

(Gewichtung: 10%)

Lizenz Idealerweise ist die Lizenz des Frameworks Open-Source und die Verwendung kostenlos.

(Gewichtung: 20%)

Mächtigkeit des NLP-Service Der Native-Language-Processing-Service muss fähig sein, Intents aus den Inputs der Benutzer herauszufiltern. Der NLP-Service muss in der Lage sein, die relevanteste Antwort auf den Input zu liefern.

(Gewichtung: 10%)

Importmöglichkeiten des Q&A-Content Der Import von Content sollte möglichst einfach sein, vorzugsweise im Markdown- oder JSON-Format.

(Gewichtung: 10%)

Zukunftssicherheit Das Framework besteht schon seit einiger Zeit und die Verwendung ist weit verbreitet. Um eine zukunftssichere Existenz voraussagen zu können, ist es wichtig, dass das Framework von einem aktiven Entwicklerteam getragen wird.

(Gewichtung: 15%)

Natural-Language-Support Idealerweise unterstützt das Framework neben Englisch noch weitere Sprachen.

(Gewichtung: 5%)

Programmiersprache Das Framework sollte eine dem Entwicklerteam bekannte Programmiersprache unterstützen. Dazu zählt C#, Java und JavaScript.

(Gewichtung: 20%)

UI-Elemente Das Design der UI-Komponenten sollte mit dem Framework möglichst einfach zu machen sein.

(Gewichtung: 10%)

Die ausgefüllte Nutzwertanalyse befindet sich im Anhang A.

3.3. Auswahl Bot-Development-Frameworks

Mittels Nutzwertanalyse haben wir sieben Bot-Development-Frameworks genauer analysiert. Als Resultat dieser Analyse haben wir eine engere Auswahl von vier verschiedenen Frameworks gewonnen, die wir genauer beschreiben.

3.3.1. Dialogflow

Kernkonzept Grundsätzlich lassen sich mit Dialogflow regelbasierte Chatbots erstellen, die von Machine Learning Gebrauch machen, um zu verstehen, was ein Benutzer schreibt. Die Hauptbestandteile eines Chatbots sind Agents, Intents, Entities und Actions. Ein Agent entspricht dem Projekt, also dem Architekturchatbot und kann als Natural-Language-Understanding Modul beschrieben werden, welche Benutzer-Requests in verwertbare Daten umwandelt. Ein Agent enthält mehrere Intents und Entities. Intents dienen dazu, aus dem Benutzer-Input, die vom Chatbot umzusetzende Action herauszufinden. Entities werden dazu verwendet, um von Inputs Parameterwerte zu extrahieren. Eine Action reagiert, wenn ein bestimmter Intent ausgelöst wird. Actions können Paramter haben.

Erster Eindruck Dialogflow sieht sehr vielversprechend aus. Dialogflow wird von Google unterstützt und wird von namhaften Kunden wie zum Beispiel Mercedes, KLM und vom Wall Street Journal verwendet. Content kann mit JSON hinzugefügt werden. Es gibt zahlreiche SDKs für die meistverbreiteten Plattformen (Android, iOS, HTML, JavaScript, Node.js, .Net) und kann mittels One-Click-Integration auf 14 verschiedene Channels deployed werden. Die Lizenz ist Apache 2.0 und Dialogflow unterstützt zahlreiche Sprachen. Für das Gestalten von UI-Elementen gibt es verschiedene Vorlagen. Es können UI-Elemente erstellt werden, die Text, Bilder und Cards (Kombination von Bild, Titel und Text) enthalten.

3.3.2. Microsoft-Bot-Framework

Kernkonzept Die Hauptkomponenten sind Channels, Bot-Connectors, Activities, Messages, und Dialoge. Der Bot-Connector verbindet den Chatbot mit einem oder mehreren Channels und

behandelt den Nachrichtenaustausch unter Verwendung von Activities. Jegliche Art von Kommunikation gilt als Activity. Messages sind ein bestimmter Typ von Activity. Eine Message kann einfach nur Text sein, Attatchments oder interaktive Elemente enthalten. Dialoge werden verwendet, um den Konversationsfluss zu organisieren. Beispiel für ein Dialog ist das Erfassen von NFRs. Dialoge sind in einem Stack organisiert. Der oberste Dialog im Stacks verarbeitet alle einkommenden Messages, bis dieser geschlossen wird oder ein anderer Dialog aufgerufen wird. Adaptiert von[18].

Erster Eindruck Das Microsoft-Bot-Framework bietet mit LUIS (Language-Understanding-Intelligence-Service) eine sehr mächtige Natural-Language-Processing-Umgebung an, welche mehrere APIs beinhaltet. Die Text-Analytics-API wird verwendet, um Schlüsselwörter, Themen und Sprache auszulesen. Mit der Bing-Spell-Check-API kann die Rechtschreibung und Grammatik geprüft werden und Namen ausgelesen werden. Die Lizenz ist MIT und somit akzeptabel. Content kann als JSON-File hinzugefügt werden. Wie auch Dialogflow, kann der Chatbot auf viele verschiedene Channels deployed werden und es werden zahlreiche Sprachen unterstützt. Das Framework bietet SDKs für .Net, Node.js und REST. Ebenfalls für das Microsoft-Bot-Framework sprechen über 4500 Stars auf Github. Das Design der UI-Elemente ist sehr channelspezifisch, jedoch bietet Web-Chat neben Text auch Cards und Attachments.

3.3.3. Wit.ai

Kernkonzept Im Wesentlichen kann Wit.ai als NLP-Service angesehen werden und weniger als Bot-Development-Framework. Wit.ai kann dazu verwendet werden, um die Message in strukturierte Daten umzuwandeln (also die Absicht des Benutzers herauszufiltern). Wie diese Daten jedoch weiterverwendet werden, ist dem Entwickler überlassen.

Erster Eindruck Mit Facebook wird das Projekt von einem namhaften Unternehmen unterstützt. Wit.ai ist lizenzfrei und überlässt dem Entwickler somit viel Freiraum, da praktisch alle Möglichkeiten erlaubt werden.

3.3.4. Botpress

Kernkonzept Botpress ist ein Code-First-Framework, welches selbst gehostet wird. Im Gegensatz zu den anderen Bot-Development-Frameworks ist Botpress modular aufgebaut. Der Chatbot ist eine reguläre Node.js Applikation und deshalb sehr flexibel. Botpress enthält nur die Logik und der Konversationsfluss des Chatbots. Botpress greift auf Incoming- und Outgoing-Middlewares zurück, um Natural-Language-Processing-Module und Channels mittels Connectors zu verbinden.

Erster Eindruck Botpress ist modular aufgebaut und kann deshalb einfach um die benötigten Komponenten erweitert werden. Auch können veraltete Komponenten einfach ausgetauscht werden. Der Content wird mittels UMM (Universal Message Markdown) hinzugefügt. Die Programmiersprache ist JavaScript. Mittels Connectors können zahlreiche Channels abgedeckt werden (Web, Facebook, Slack, Skype, Kik). Auf Github ist Botpress sehr aktiv und weist über 3500 Stars auf. Die Dokumentation ist verständlich und scheint auf den ersten Blick alles abzudecken. Die Lizenz ist GNU Affero General Public License v3.0. Sehr überzeugende Argumente für Botpress sind die modulare Architektur und der Code-First-Ansatz.

Möglichkeiten Botpress kann auch in Kombination mit Dialogflow und Wit.ai als NLP-Middleware verwendet werden. Botpress selbst stellt dafür Connectors zur Verfügung. Der Vorteil ist, dass wir den Konversationsfluss und den Content mit einer Programmiersprache definieren können und der Chatbot den NLP-Service von einem der beiden Frameworks verwendet. Der Konversationsfluss und der Content wäre somit von den beiden Frameworks losgelöst und der NLP-Service wäre einfach auszuwechseln.

3.4. Prototypevaluation

Ziel der Prototypevaluation war es, dass wir durch das Erstellen von zwei Prototypen, das passendste Framework, für die Realisierung des ArchBots finden. Beide Prototypen implementierten dieselbe User-Story, welche als Vergleichsbasis dient, um die Frameworks zu bewerten. Ebenfalls zur Gewährleistung der Vergleichsfähigkeit wurde ein Konversationsfluss definiert, an welchen sich die Umsetzung der Prototypen orientieren. Das Szenario für die Evaluation ist im Umfang beschränkt, was die Entwicklung der Prototypen beschleunigt. Es weist jedoch die relevanten Kernpunkte auf.

Das Szenario ist an die Aufgabenstellung angelehnt. Dies ermöglicht es uns, mit dem aus der Prototypevaluation hervorgehenden Prototypen weiter zu arbeiten. Die für die Evaluation des Frameworks wichtigsten funktionalen Anforderungen werden vom Prototypevaluatiosszenario abgedeckt. Das Szenario, welches in der Evaluation verwendet wird, ist ein Teilbereich der funktionalen Anforderungen «Informationssammlung».

3.4.1. Kriterien

Wichtige Kriterien zur Berücksichtigung bei der Entscheidung des Prototyps:

- Konnten alle nicht-funktionalen Anforderungen erfüllt werden?
- Konnte die gewünschte Funktionalität umgesetzt werden und ist es denkbar, die restlichen funktionalen Anforderungen zu integrieren?
- Wie einfach war es, den Prototypen umzusetzen? Hierbei handelt es sich um ein subjektives Kriterium, welches durch Austausch unter dem Entwicklerteam bewertet wird.
- Waren genügend Informationen zum Framework vorhanden? Unter Berücksichtigung der offiziellen Dokumentation, sowie der Informationen die durch die Community zur Verfügung gestellt werden.
- Gibt es eine entsprechende Umgebung zum Framework? Können Hygienefaktoren einfach umgesetzt werden und gibt es Third-Party-Components mit welchen das Framework erweiterbar ist?

3.4.2. Vorgehen

Jedes Teammitglied entwickelt einen Prototypen, basierend auf einem anderen Framework. Sind die Prototypen fertig, werden die Prototypen anhand der Kriterien bewertet. Das ermöglicht es uns zu entscheiden, welcher Prototyp weiterentwickelt wird.

3.4.3. Prototypen

Es wurden zwei Prototypen umgesetzt:

Prototyp 1: Umsetzung auf Basis des Microsoft-Bot-Framework mit LUIS als NLP-Service.

Prototyp 2: Umsetzung aus Kombination von Botpress und Dialogflow und/oder Wit.ai als NLP-Service.

3.4.4. User-Story

Als Softwarearchitekt möchte ich vom Chatbot durch ein Gespräch geführt werden, in dem nicht-funktionale Anforderungen beleuchtet und gesammelt werden, sodass ein Administrator meine Antworten auslesen kann.

3.4.5. Funktionalität

Folgende Funktionen müssen mit den Prototypen umsetzbar sein:

- Der Chatbot begrüsst den Benutzer (Onboarding) und instruiert diesen.
- Der Benutzer kann Informationen über nicht-funktionale Anforderungen verlangen und erhält als Antwort entsprechende Literatur.
- Der Benutzer kann NFRs hinzufügen.
- Der Benutzer kann NFRS löschen.
- Der Benutzer kann sich die gesammelten NFRs anzeigen lassen.

3.5. User-Experience

3.5.1. Nachrichtenlänge

Der Chatbot sollte möglichst kurze Aussagen machen, damit der Benutzer nicht abgeschreckt wird und den Text gar nicht liest. Idealerweise besteht eine Nachricht aus zwei Zeilen Text auf einem

mobilen Gerät. Dies entspricht in etwa 63 Zeichen. Maximal sollte eine Nachricht aus drei Zeilen (90 Zeichen) bestehen. Kann das nicht eingehalten werden, muss die Nachricht auf mehrere Nachrichten aufgeteilt gesendet werden. Diese sollten jedoch insgesamt 140 Zeichen nicht überschreiten, bis zur nächsten Aktion des Benutzers. Ebenfalls ist darauf zu achten, dass der Benutzer nicht gezwungen ist, im Chat zu scrollen, um die Informationen zu sehen, welche für seine nächste Interaktion notwendig sind[15].

3.5.2. Navigation

Zwei wichtige Kernpunkte, welche für sonstige Applikationen im Bezug auf die Navigation wichtig sind, gelten auch für Chatbots.

- Dem Benutzer muss bewusst sein, an welchem Ort er sich im Chatbot befindet. Das heisst, der Chatbot muss dem Benutzer mitteilen, wo er sich befindet bzw. welche Aktionen er benutzen kann. Das kann der Chatbot als Aussage formulieren oder es muss in das UI des Chats eingebaut werden.

- Dem Benutzer muss einen Weg zur Navigation durch die verschiedenen Dialoge und Aktionen des Chatbots gegeben werden. Dies kann durch Aussagen des Benutzers gegenüber dem Chatbot umgesetzt werden. Eine weitere Möglichkeit ist das Integrieren von UI-Elementen, wie zum Beispiel ein Zurück- und Home-Button[15].

3.5.3. Korrekturmöglichkeit

Der Chatbot muss eine Möglichkeit anbieten, die es dem Benutzer ermöglicht, Fehler zu korrigieren[15].

3.5.4. Hilfestellung

Um den Benutzer zu unterstützen, soll der Benutzer jederzeit Zugriff auf möglichst präzise Hilfe haben. Diese kann beispielsweise durch eine Nachricht wie «I need Help!»aufgerufen werden.

3.5.5. Kommunikation

Da zwischen Chatbot und Benutzer eine Konversation stattfindet, können bekannte Theorien der Kommunikation auch auf Chatbots angewendet werden.

3.5.6. Verständlichkeit

Der Benutzer soll in möglichst kurzen und verständlichen Nachrichten mit dem Benutzer kommunizieren. Versteht der Benutzer die Nachricht nicht, sollte der Chatbot eine Nachricht mit einer alternativen Formulierung senden.

3.5.7. Nachfragen

Um sicherzustellen, dass das Gespräch reibungslos verläuft und keine Missverständnisse entstehen, soll der Chatbot den Benutzer kurz wiederholen. Dies gibt dem Benutzer die Möglichkeit, den Chatbot bei einem Missverständnis zu korrigieren.

3.6. Testmetriken

Um die Prototypen und anschliessend auch die Alpha-, Beta-, Final-Releases zu testen, definieren wir Testmetriken. Die Testmetriken umfassen hauptsächlich Usability- und Performance-Faktoren.

Anzahl Schritte in einer Konversation Durchschnittliche Anzahl Schritte pro Konversation. Als Konversationsschritte zählt ein Request an den Chatbot und die dazugehörige Antwort. Wenn also z.B. der Benutzer «Hallo» schreibt und der Chatbot mit «Guten Tag» antwortet, zählt das als ein Konversationsschritt.

Dauer einer Konversation Durchschnittliche Dauer einer Konversation.

Anzahl Confusion Triggers Misst, wie oft der Chatbot den Benutzerinput nicht interpretieren kann. Messbar, in dem gezählt wird, wie oft der Chatbot mit «Ich verstehe Sie nicht.» reagiert.

Antwortzeit Misst die durchschnittliche Antwortzeit des Chatbots. Diese sollte sehr tief sein (etwa eine Sekunde).

Konversionsrate Misst, wieviele Benutzer dazu bereit sind, dem Chatbot ihre E-Mail-Adresse zu geben.

Anteil Benutzer die aufgeben Prozentualer Anteil Benutzer, die eine Konversation frühzeitig abbrechen, ohne ein Resultat zu bekommen.

3.7. Prototypen

3.7.1. Botpress in Kombination mit Dialogflow und Wit.ai

Installation und Einrichtung Um die NLP-Services Dialogflow und Wit.ai zu verwenden, musste jeweils ein Account erstellt werden. Beide Plattformen sind übersichtlich aufgebaut. Die Installation von Botpress war einfach und verlief problemlos. Nach dem Start von Botpress kann auf einen Webserver zugegriffen werden, in dem der Chatbot konfiguriert werden kann. Die Web-Oberfläche bietet eine gute Übersicht und ermöglicht einen schnellen Einstieg in das Framework. Man erkennt die Absicht der Entwickler mit Botpress eine Art WordPress für Chatbots zu bieten.

Evaluation Über die Web-Oberfläche konnten problemlos die verschiedenen Module, NLP-Services, Channels und Datenbanken, integriert und konfiguriert werden. Die Web-Oberfläche stellt nur ein Hilfswerkzeug dar, alle Module können auch über die Kommandozeile integriert und über Konfigurationsfiles konfiguriert werden.

Das Trainieren von Dialogflow und Wit.ai auf das Erkennen von Intents und Entitys war einfach. Dialogflow ist benutzerfreundlicher, weil die Plattform besser strukturiert ist und viele Beispiele und eine gute Dokumentation zur Verfügung stehen.

Das Universal-Message-Markdown, welches verwendet wird, um die Nachrichten vom Chatbot an den Benutzer zu definieren war praktisch. Es konnten problemlos, die für das Prototypevaluationsszenario, benötigten Funktionen integriert werden. Durch Threads wird das Eintauchen in verschiedene Dialoge des Konversationsflusses ermöglicht. Unter der Verwendung von NLP-Services war die Umsetzung des Konversationsflusses nicht möglich, weil NLP-basierte Trigger noch nicht unterstützt wurden. Hier zeigte sich, dass die Dokumentation unvollständig ist. Selbst durch Recherche im Internet und Nachfragen im Botpress-Slack-Channel konnten keine weiteren Informationen mehr beschafft werden.

Botpress bietet standardmässig einen Web-Chat als Channel an. Dieser konnte problemlos als Modul integriert und konfiguriert werden. Im Verlauf der Umsetzung stellte sich jedoch heraus, dass dieser Web-Chat noch nicht ausgereift ist. So können zum aktuellen Zeitpunkt weder Carousels, noch Bilder an den Benutzer gesendet werden.

Die Verwendung SQLite3 als Built-In-Datenbank funktionierte problemlos und es kann davon ausgegangen werden, dass die Verwendung von Postgres, welches als externe Datenbank empfohlen wird, ebenfalls reibungslos abläuft.

Fazit Unter der Verwendung von Botpress in Kombination mit einem NLP-Service konnte das Prototypevaluationsszenario nicht umgesetzt werden. Weshalb das Umsetzen der Applikation auf Basis dieses Prototyps nicht empfehlenswert ist. Botpress verfolgt einen guten Ansatz, ist jedoch noch nicht ausgereift. Es fehlen wichtige Funktionen, die Dokumentation ist mangelhaft, und die es gibt keine aktive Botpress-Community. Dialogflow, wie auch Wit.ai stellen sich als gute NLP-Services heraus. Wit.ai kann bezüglich Benutzerfreundlichkeit allerdings nicht ganz mit Dialogflow mithalten.

3.7.2. Microsoft-Bot-Framework

Installation und Einrichtung Der Einstieg in das Microsoft-Bot-Framework war aufgrund von zahlreichen Beispielprojekten unkompliziert. Um sich mit dem Chatbot zu verbinden, mussten wir zusätzlich den Microsoft-Bot-Emulator installieren. Der Server war einfach zu starten. Sobald dieser lief, konnte dem Emulator die Adresse und den Port des Servers übergeben werden und schon war man verbunden. Für LUIS haben wir einen Account registriert und eine LUIS-App erstellt.

Evaluation Am Anfang gab es Probleme beim Verbinden mit LUIS. So haben wir zuerst die Funktionalität (Onboarding) welche keinen Native-Language-Processing-Service benötigt umgesetzt. Anschliessend gelang es doch, den Chatbot mit LUIS zu verbinden und so konnte auch der Rest des Prototypevaluationsszenario (Hinzufügen/Löschen und Anzeigen von NFRs)

implementiert werden.

Im Bot-Development-Framework werden einzelne Steps in zusammenhängende Dialoge gegliedert, auf denen Trigger registriert werden können. Sobald der Benutzer eines dieser Schlüsselwörter eingibt und den Trigger auslöst, wird dieser Dialog gestartet. Es war jedoch auch möglich aus einem bestimmten Dialog, andere Dialoge zu starten.

Mit LUIS können aus den Benutzerinputs Intents ausgelesen werden. Grundsätzlich werden in der LUIS-App Intents definiert und dann dazugehörige Beispielsätze hinzugefügt. Auch gibt es vorgefertigte Intents, die hinzugefügt werden können. Zusätzlich werden auch Entities definiert, die eine Klasse von ähnlichen Objekten repräsentiert (in unserem Prototypevaluationsszenario zum Beispiel «Name des NFRs». Anschliessend wird der Chatbot trainiert, indem LUIS die Beispielsätze verallgemeinert und Code generiert, der relevante Intents und Entities erkennt. Diese Intents haben wir dann als Trigger für die Dialoge festgelegt.

Für den Prototypen haben wir noch keine Datenbank eingebunden. Da eine Datenbankanbindung für einen Node.js Server ohne Probleme umsetzbar sein sollte, haben wir die NFRs und die weiteren Daten in der aktuellen Session abgespeichert.

Fazit Für den Prototypen haben wir uns für das Node.js Software-Development-Kit (SDK) entschieden, da wir nicht mit Visual Studio auf einer Virtual Machine arbeiten wollten. Microsoft stellt einen Emulator zur Verfügung, um den Chatbot auszuführen und zu debuggen.

Die gewünschten Funktionen, welche im Prototypevaluationsszenario definiert wurden, konnten vollumfänglich umgesetzt werden und es ist deshalb anzunehmen, dass die restlichen Funktionalitäten ebenfalls umsetzbar sind.

LUIS bedarf etwas Einarbeitungszeit, da es wenig intuitiv ist und ein theoretisches Verständnis von Native Language Processing voraussetzt. Sobald man sich eingearbeitet hat und LUIS mit dem Chatbot verbunden hat, funktioniert es allerdings gut und liefert brauchbare Ergebnisse beim Erkennen der Intents.

Nach etwas Einarbeitungszeit und Problemen mit LUIS ging die Entwicklung relativ schnell voran. Die verschiedenen UI-Komponenten können einfach umgesetzt werden. Als einziger schwerwiegender Nachteil konnte die starke Bindung von Content und Konversationssfluss identifiziert werden.

3.7.3. Entscheidung

Wir haben uns entschlossen, das Projekt mit Microsoft-Bot-Framework weiterzuführen. Neben den offensichtlichen Vorteilen, wie der Emulator mit den vordefinierten UI-Elementen, überragten auch Kriterien, welche wir bei der Nutzwertanalyse nicht evaluieren konnten, wie Qualität der Dokumentation und Menge an externen Ressourcen. Bei Botpress funktionierte das Zusammenspiel der einzelnen Komponenten nicht wie angegeben. Insbesondere gab es einen Konflikt zwischen dem Verwenden von NLP-basierten Triggern und einem sinnvollen Konversationsfluss.

4. Architekturdokumentation des ArchBots

In diesem Kapitel behandeln wir die logische Architektur und das Deplyoment des ArchBots.

4.1. Logische Architektur

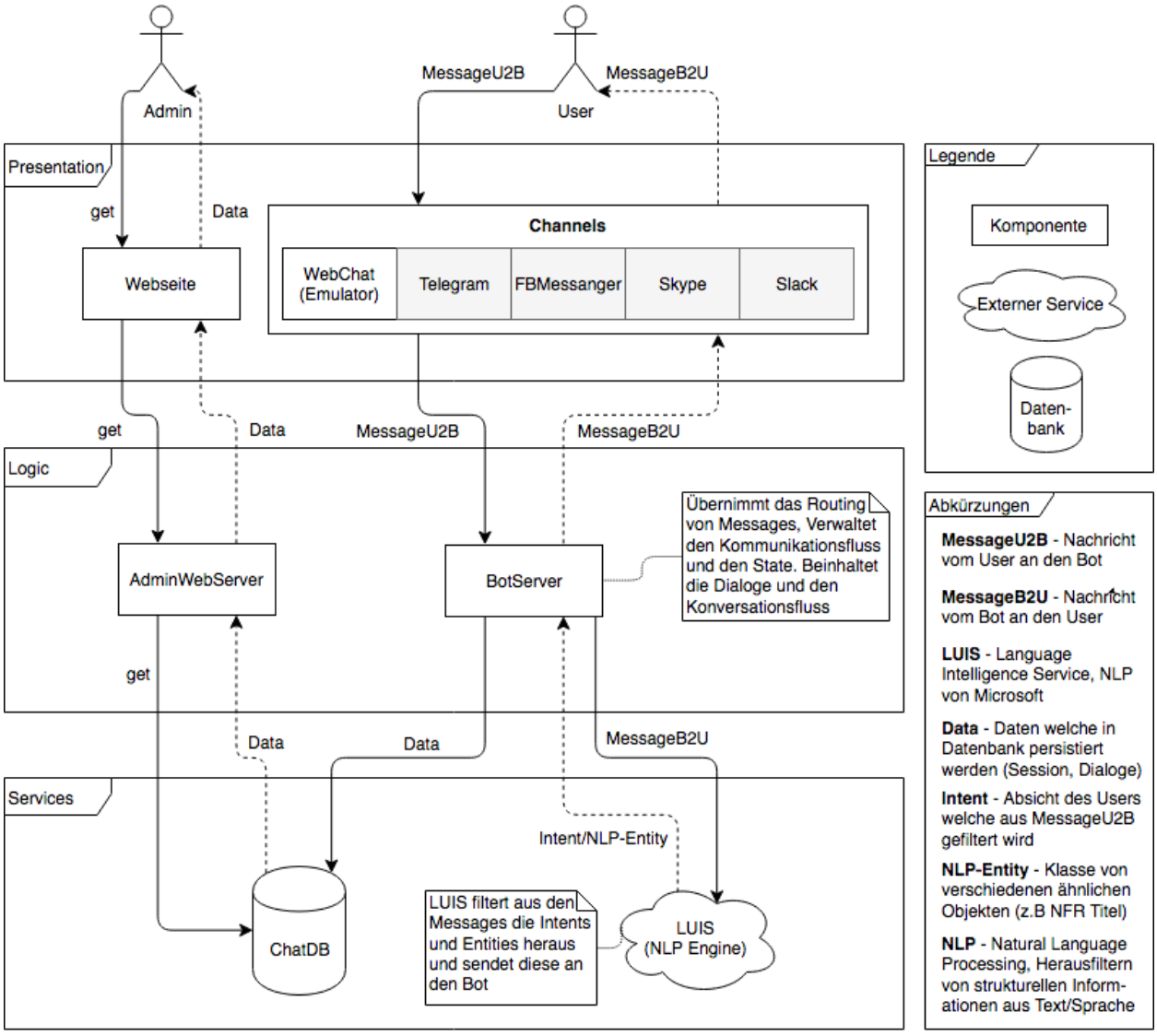


Abbildung 4.1.: Logische Architektur

Wie aus der Abbildung 4.1 ersichtlich ist, teilten wir die Architektur in drei Schichten auf. Die Präsentationsschicht ist die Schicht über die der Benutzer mit dem ArchBot kommuniziert. Diese

Schicht umfasst im Wesentlichen die verschiedenen möglichen Channels und die Webseite für den Admin um die Konversationen auszuwerten. Für unseren Chatbot beschränken wir uns auf den Microsoft-Bot-Emulator, welcher auf den Web-Chat-Channel basiert. Die vom Benutzer in einen Channel eingegebene Nachricht geht von der Präsentationsschicht weiter in die logische Schicht. In der logischen Schicht befindet sich der BotServer, der das Routing übernimmt und die Dialoge und den Konversationsfluss beinhaltet. Zusätzlich ist in dieser Schicht der AdminWebServer zu finden, der eine Schnittstelle zwischen der Admin-Webseite und der ChatDB ist. Die unterste Schicht ist die Service Schicht. Dieser Schicht ist sowohl die ChatDB, die dazu verwendet wird um die Konversationen zwischen Benutzer und Bot zu speichern, als auch der externe NLP-Service LUIS, welche zu den Nachrichten passende Intents herausfiltert, zuzuordnen.

4.1.1. Komponenten

Benutzer (User) Der Benutzer interagiert mit dem Bot. Über einen Channel sendet er Nachrichten (MessageU2B) an den Bot und erhält Nachrichten (MessageB2U) zurück.

Admin Ein Admin kann die gespeicherten Konversationen auslesen, um weiter zu verarbeiten.

Webseite Dem Admin steht eine Webseite zur Verfügung, auf die er alle Konversationen ansehen kann. Auch kann er sich alle Antworten zu den einzelnen Fragen anzeigen lassen, um gegebenenfalls unverständlich formulierte Fragen zu finden.

Channels Der Chatbot kann auf verschiedene Channels bereitgestellt werden. So kann er in bestehende Messaging Systeme eingebunden werden. Für unsere Arbeit beschränken wir uns jedoch auf den Bot Framework Emulator, welcher auf den WebChat basiert.

AdminWebServer Der AdminWebServer stellt eine Schnittstelle zwischen der Admin-Webseite und der ChatDB dar.

BotServer Übernimmt das Routing von Messages vom ArchBot zum Channel und zurück. Verwaltet den Kommunikationsfluss und den State der Applikation. Der BotServer übernimmt auch das Session Handling, beinhaltet die Dialoge und den Konversationsfluss. Diese Komponente umfasst die allgemeinen Komponenten Action, Responder und die Chatbot-Applikation, welche in der allgemeinen Architektur beschrieben werden.

ChatDB In der ChatDB werden die einzelnen Sessions abgespeichert, um die Konversationen auswerten zu können.

LUIS Der NLP-Service LUIS filtert aus den Messages die Intents und Entities heraus und sendet diese an den Bot. Es handelt sich hierbei um einen externen Service, welcher von Microsoft zur Verfügung gestellt wird. LUIS entspricht dem Parser, welcher im Unterkapitel Architektur in der Vorstudie beschrieben wurde.

4.1.2. Beschreibung des Datenflusses

Der Benutzer sendet eine Nachricht über den Web-Chat-Channel. Die Nachricht wird über eine mit HTTPS gesicherte REST-Schnittstelle an den BotServer übertragen. Wenn die Nachricht einen

fest eingebauten Dialogtrigger beinhaltet, wird der entsprechende Dialog gestartet. Es kann jedoch auch sein, dass die Nachricht des Benutzers eine Antwort auf eine Frage des Chatbots ist. Dann wird einfach im aktuellen Dialog fortgefahren. Jede Nachricht wird an den NLP-Service gesendet, welche versucht, den Intent aus der Nachricht zu filtern. Beinhaltet die Nachricht einen NLP-basierten Trigger, wird der entsprechende Intent an den ArchBot zurückgesendet und so startet ebenfalls der entsprechende Dialog. Die Antwort des Chatbots wird über den umgekehrten Weg zurück an den Benutzer gesendet. Beendet der Benutzer die Konversation, werden alle Daten in die Datenbank gespeichert. Der Chatbot hat eine Admin-Funktion eingebaut, mit der ein Benutzer die gespeicherten Konversationen abrufen kann. Dazu wird ein bestimmter Dialog ausgelöst, der dem Benutzer eine URL zur Adminwebseite sendet.

4.2. Deployment

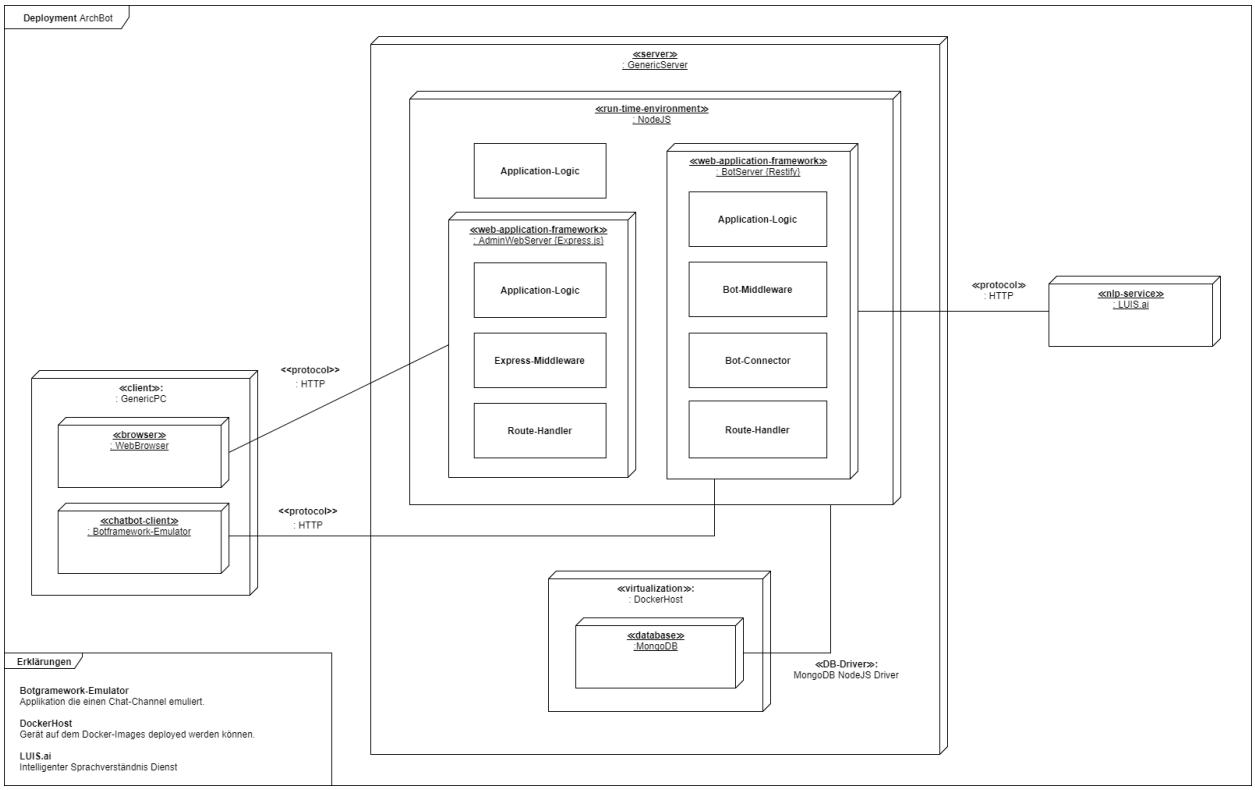


Abbildung 4.2.: Deploymentdiagramm des ArchBots

Wie in der Abbildung 4.2 zu sehen ist, setzt sich der ArchBot aus mehreren Softwareartefakten und -instanzen zusammen, die über verschiedene Nodes und Komponenten verteilt sind. Es kommen zwei physikalische Device-Nodes und ein Web-Service-Node zum Einsatz. Die drei grundlegenden Nodes sind der «client»-Node, mit welchem der Benutzer auf den ArchBot zugreift, der «server»-Node, welcher den ArchBot hostet, und der Web-Service-Node «web-nlp-service», welcher die Nachrichten der Benutzer verarbeitet.

4.2.1. Beschreibung der Nodes für das Deployment

client Stellt einen generischen Computer des Chatbot-Benutzers oder des Administrators dar.

chatbot client Der Chatbot-Client ist der MS Bot Framework Emulator. Der Emulator kommuniziert über HTTP mit dem «BotServer».

browser Für den Zugriff auf die Administrationsoberfläche, um die vom Chatbot erhobenen Daten auszulesen, wird ein Webbrowser benötigt, welcher über HTTP mit dem «AdminWebServer» kommuniziert.

server Der Server ist der physikalische Node, auf dem die Hauptkomponenten des ArchBots liegen. Er bietet die Umgebung für das Hosting des Bot-Servers, des «AdminWebServer» und einer Datenbank.

run-time environment Als «run-time environment» verwenden wir einen Node.JS Server, welches die Grundlage für den «BotServer» und den «AdminWebServer» bietet. Mittels Verbindung über den DB-Driver MongoDB Node.js Driver auf die virtualisierte Datenbank, bietet es den Servern die Möglichkeit Daten zu speichern und auszulesen.

web-application-framework Die Nodes bieten die Grundlage und das Umfeld für das Erstellen von Web-Applikationen. Es sind zwei verschiedene Instanzen im Einsatz, eine für den BotServer und eine für den «AdminWebServer».

BotServer Der «BotServer» ist zuständig für das Ausführen des ArchBot. Durch das Bereitstellen eines Web-Endpunktes, der auf Restify basiert, wird die Kommunikation mit dem Chatbot-Client ermöglicht. Eine Verbindung zum «web-nlp-service» LUIS gibt dem ArchBot die Fähigkeit, Nachrichten des Benutzers durch künstliche Intelligenz zu verstehen.

AdminWebServer Um dem Administrator des ArchBots Zugriff auf die in der Datenbank gespeicherten Daten zu gewähren, stellt der «AdminWebServer» mehrere Web-Endpunkte bereit, die mittels HTTP erreicht werden können. Der AdminWebServer basiert auf einem Node.js Express Server.

virtualisation Für ein einfacheres Deployment der Datenbank wird Docker als Host eingesetzt, welcher die Verwendung von Docker-Images ermöglicht.

database Für die Instanzierung der Datenbank wird ein Docker-Image von MongoDB verwendet. Die Datenbank stellt dem «run-time environment» den Zugriff auf die Daten bereit und ermöglicht den «web applikation framework»-Instanzen, Daten auszulesen und zu schreiben.

web-nlp-service Dieser Node stellt LUIS dar und ermöglicht dem «BotServer» somit, die Nachrichten des Benutzers zu parsen und verstehen.

4.2.2. Beschreibung der Komponenten

Node.js

Application Logic Diese Komponente beinhaltet das Set-up und Instanzierung der «web-application-frameworks» und der Datenbankverbindung. Ebenfalls stellt die Komponente Funktionen zur Verfügung, um mit der Datenbank zu interagieren.

BotServer

Application-Logic Diese Komponente beinhaltet die Programmlogik des ArchBots.

Bot-Middleware Alle Nachrichten von und zum «BotServer» fliessen durch diese Komponente. Dadurch wird das Parsen der Nachrichten durch den «web-nlp-service» ermöglicht.

Bot-Connector Der Verbindungsaufbau zwischen dem «BotServer» und dem «Chatbot-Client» wird von dieser Komponente übernommen.

Route-Handler Der «Route-Handler» wird dazu verwendet, um einen Web-Endpoint bereitzustellen und die Anfragen an den Bot-Connector weiterzuleiten.

AdminWebServer

Application-Logic Diese Komponente beinhaltet die Programmlogik des Adminbereichs. Sie liest Daten aus der Datenbank aus und bereitet diese auf.

Express-Middleware Diese Komponente stellt das Bindeglied zwischen den Web-Endpoints und der Programmlogik dar.

Route-Handler Um verschiedene Web-Endpoints bereitzustellen wird diese Komponente benötigt.

5. Umsetzung des ArchBots

Dieses Kapitel befasst sich mit den Entscheidungen, die wir getroffen haben und wie wir den ArchBot implementiert und getestet haben.

5.1. Framework-, Software- und Serviceentscheidungen

Im Verlauf der Arbeit mussten wir uns für verschiedene Frameworks, Softwares und Services entscheiden. Nachfolgend sind die getroffenen Entscheidungen für die einzelnen Teile des ArchBots dokumentiert:

BotServer Für die Implementierung des «BotServers», wurde das Microsoft-Bot-Framework verwendet, welches aus der Prototypevaluation hervor ging. Das Microsoft-Bot-Framework wird standardmässig mit Restify ausgeliefert, könnte aber durchaus ebenfalls mit Express.js betrieben werden. Restify ist ein Web-Service-Framework, welches dazu verwendet wird, um semantisch korrekte RESTful-Web-Services zu entwickeln. Nach einem erfolglosen Versuch das Microsoft-Bot-Framework mit Express.js zu betreiben, fiel die Entscheidung auf Restify.

AdminWebServer Zur Implementierung des «AdminWebServers» wurde Express.js verwendet. Prinzipiell hätten wir hierfür auch Restify als Framework verwenden können, welches bereits für den «BotServer» Verwendung findet, allerdings haben wir uns für Express.js entschieden, da wir bereits Erfahrung mit der Verwendung von diesem Framework haben.

database Zur Implementierung der Datenbank haben wir die Document-Oriented-Database MongoDB verwendet, weil der ArchBot keine komplexe Datenstruktur und Operationen erfordert. Es müssen nur Daten zu den jeweiligen Sessions gespeichert werden. Die Verwendung einer Document-Oriented-Database bietet den Vorteil, dass JSON-Objekte direkt und einfach in die Datenbank geschrieben und ausgelesen werden können, ohne der Verwendung eines ORM-Layers. Weil das Team bereits Erfahrung in der Verwendung von MongoDB als Document-Oriented-Database hat, war diese Entscheidung einfach zu treffen. Wie im Abschnitt 4.2 beschrieben, wird die Datenbank in einem Docker-Host betrieben, was die Installation beträchtlich vereinfacht. Mittels Startparameter kann dem Docker-Image der MongoDB ein lokales Verzeichnis mitgegeben werden, in dem die Datenbank persistiert wird.

client Als Client diente uns der Bot-Framework-Emulator von Microsoft. Ursprünglich war geplant, den Bot auf einer eigenen Webseite zu integrieren. Zur Reduktion des Aufwandes haben wir uns im Verlauf des Projekts dafür entschieden, den Emulator zu verwenden. Der Bot-Framework-Emulator wurde entwickelt, um «Bot-Channel»-unabhängig zu testen und besitzt zu diesem Zweck entsprechende Funktionen. Unter anderem ermöglicht der Bot-Framework-Emulator einen Einblick in die Metadaten der Nachrichten und erleichtert durch so das Debugging. Ein Screenshot des Emulators ist in der Abbildung 5.1 zu sehen.

nlp-service Für die Implementierung des «nlp-service», verwenden wir LUIS (Language-Understanding-Intelligent-Service), den NLP-Service von Microsoft, was ebenfalls aus der Prototypenevaluation hervor ging. LUIS als «nlp-service» bietet zusätzlich den Vorteil, das die Konfiguration

mittels JSON-File sehr einfach exportiert und importiert werden kann.

Testing Für das Testen des ArchBots haben wir uns für das «Bot Tester for Bot Builder Framework» entschieden, welches auf der Assertion-Library «Chai» und dem JavaScript-Testing-Framework «Mocha» aufbaut. Dieses Framework stammt von einem Drittanbieter, erleichtert jedoch das Schreiben von Unit-Tests stark.

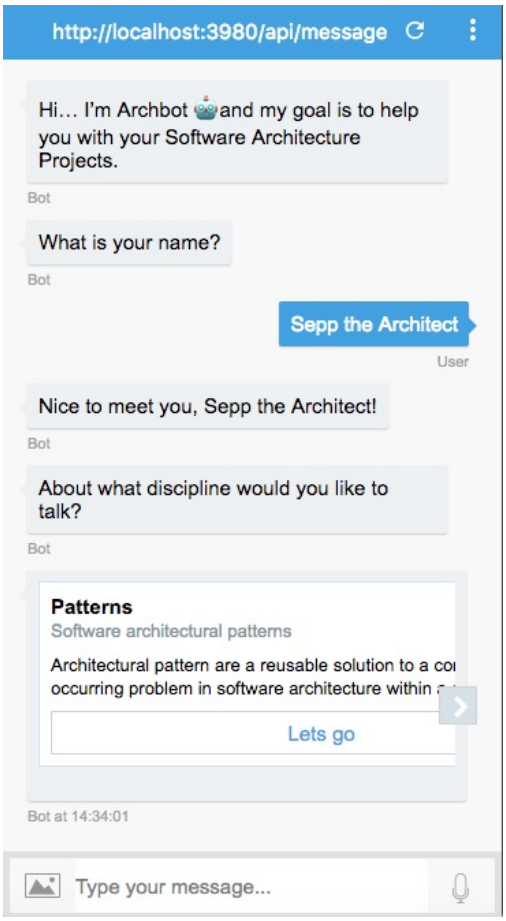


Abbildung 5.1.: Screenshot des Emulators mit Konversation zwischen dem ArchBot und dem Benutzer

5.2. BotServer

Die Implementierung des BotServers basiert auf dem Microsoft-Bot-Framework in Kombination mit Restify, und besitzt Zugriff auf die Datenbank, in welcher die Konversationen gespeichert werden. Ausserdem besitzt der «BotServer» eine Anbindung an den NLP-Service LUIS, um die Nachrichten des Benutzers zu verstehen.

5.2.1. Session-Handling

Jede Verbindung zwischen dem ArchBot und dem «Client» muss zuerst aufgebaut werden. In diesem Schritt wird der Verbindung eine Session mit einer zugehörigen SessionID zugewiesen. Die zugewiesene Session erlaubt es dem ArchBot, den Benutzer zu identifizieren. Intern besitzt der «BotServer» ein «Session-Objekt», in dem sämtliche Daten der Session gespeichert werden. Das sind beispielsweise alle erreichbaren Dialoge und der Zustand der Session.

Aufbau der Chat-Session zwischen dem ArchBot und dem «Client»

Der «BotServer» stellt mittels Restify den Endpunkt «/api/messages» zur Verfügung, mit welchem sich der «Client» verbinden kann. Hinter diesem Endpunkt befindet sich der «Bot-Connector» welcher, mit dem «Route-Handler» den «Client» auf einen neuen Port umleitet und ihm unter diesem Port den Bot bereitstellt. In diesem Schritt weist der «Bot-Connector» dem «Client» ebenfalls eine Session zu. Nach dem erfolgreichen Aufbau der Verbindung kann die Konversation zwischen dem ArchBot und dem «Client» beginnen.

Time-Out der Session

Meldet sich der Benutzer für eine gewisse Zeit nicht, fragt der ArchBot den Benutzer mit «Are you there?», ob dieser noch anwesend ist. Reagiert der Benutzer eine gewisse Zeit lang nicht auf diese Frage, beendet der ArchBot die Session mit der Nachricht «Im ending the conversation since youve been inactive too long. Hope to see you soon.» und speichert die bereits erfassten Daten in die Datenbank.

Diese Funktion wurde in der «Bot-Middleware» definiert. Wie bereits im Unterabschnitt 4.1.2 beschrieben, werden alle Nachrichten, die über den «Client» an den ArchBot übertragen werden, durch die Komponente «Bot-Middleware» gesendet. Die Verwendung der Bot-Middleware ermöglicht es, Nachrichten losgelöst von der «Application-Logic» zu verarbeiten und findet ebenfalls Einsatz beim Weiterleiten der Nachrichten an den NLP-Service.

5.2.2. Implementierung des Konversationsflusses

Die Konversation richtet sich an den Konversationsfluss, den wir in Zusammenarbeit mit Prof. Dr. Olaf Zimmermann erarbeiteten. Der Konversationsfluss ist als Diagramm im Anhang B zu finden. Beim Erstellen des Konversationsflusses achteten wir auf die Punkte, die wir in der Vorstudie im Abschnitt 3.1.4 definiert haben.

Die Implementierung des Konversationsflusses baut auf der vom Microsoft-Bot-Frameworks vorgegebenen Struktur auf, welche bereits in der Vorstudie im Unterabschnitt 3.7.2 beschrieben ist.

Nachfolgend sind die wichtigsten Elemente aufgelistet, welche benutzt wurden, um den Konversationsfluss zu erstellen.

Dialoge sind Teilbereiche der Konversation, in welchen der Gesprächsablauf vom Chatbot vorgegeben ist. Ein Dialog besteht aus mehreren Schritten, die der Reihe nach ablaufen und zum Beispiel Fragen, Aussagen oder andere UI Elemente enthalten können.

Dialogstack ist ein Stack aus Dialogen, der denselben Regeln folgt wie die gleichnamige Datenstruktur. Der Dialog an oberster Position des Stacks ist aktiv. Dialoge können dem Dialogstack hinzugefügt, entfernt oder ersetzt werden. Wird ein neuer Dialog zum Stack hinzugefügt, speichert der abgelöste Dialog seinen Zustand innerhalb des Dialoges. Somit kann beispielsweise ein Hilfedialog von einem Dialog aus aufgerufen werden. Nach dem Beenden des Hilfedialogs, springt der Bot wieder zurück auf den Dialog, aus welchem der Hilfedialog aufgerufen wurde.

Actions sind Aktionen, die an den Dialog gebunden sind. Mit diesen Actions kann der Ablauf des Chatbots gesteuert werden. Diese Actions werden durch Nachrichten des Benutzers oder vom Chatbot direkt angesprochen. Es können durchaus mehrere «Matches» für eine Action in einem Array definiert werden, um beispielsweise bei Ausfall des NLP-Services den Bot trotzdem benutzen zu können. Es sind verschiedene Actions vordefiniert, wie beispielsweise die «triggerAction», welche den Dialog aufruft, wenn die Nachricht des Benutzers die festgelegten Matches enthält. Ebenfalls kann eine «cancelAction» definiert werden, mit welcher der Dialog beendet werden kann.

Im Listing 5.1 ist zu sehen, dass zwei «Matches» für die «triggerAction» definiert sind. Der erste «Match» ist ein Regex, der auf die Nachricht «Start» des Benutzers reagiert. Der zweite Match ist der Name des Intent, welcher im NLP-Service hinterlegt ist. Anzumerken ist noch, dass Dialoge auch proaktiv starten können, wie der Begrüssungsdialog, welcher bei einem erfolgreichen Verbindungsaufbau aktiviert wird.

In Listing 5.1 wird der Aufbau und Ablauf eines Dialogs ersichtlich. Der Benutzer startet den Dialog mit «Starte Beispiel Dialog» oder über den NLP-basierten «Match», und fügt ihn somit zum Dialogstack hinzu. Die erste Funktion bzw. der erste Schritt des Dialogs wird ausgeführt und der ArchBot sendet «Wie heisst du?» an den Benutzer. Der Benutzer antwortet mit seinem Namen. Die Antwort des Benutzers wird der zweiten Funktion als Parameter mitgegeben. Anschliessend beendet der Bot den Dialog mit «Freut mich, Benutzername!» und entfernt den Dialog vom Stack.

```
function setDialog(bot) {
 bot.dialog('BeispielDialog', [
 function (session, args, next) {
  builder.Prompts.text(session, 'Wie heisst du?')
  },
  function (session, results) {
   session.endDialog('Freut mich, ${results.response}!');
  }
 ]).triggerAction({
  matches: [/^Start/i, 'Example.Start'],
 }).cancelAction('cancelDialog', 'Abbruch des Dialogs.',{
  matches: [/^(cancel|nevermind)/i, 'Utilities.Cancel'],
 });
}
```

Listing 5.1: Codebeispiel eines kurzen Dialogs

Onboarding

Verbindet sich ein «Client» mit dem ArchBot, startet der ArchBot proaktiv das Onboarding mit dem «greetings»-Dialog. In diesem Dialog stellt sich der ArchBot vor und fragt nach dem Namen des Benutzers, um das Gespräch persönlicher zu gestalten.

Kernfunktionalität

Die Implementation des ArchBot umfasst zwei Themenbereiche, in denen er dem Benutzer bei einem Softwarearchitekturprojekt helfen kann. Ebenfalls umgesetzt wurde das Erheben und Speichern von Daten des Gesprächs, um diese dem Softwarearchitekten für einen späteren Kontakt mit dem Kunden bereitzustellen.

NFR Der ArchBot besitzt die Fähigkeit, mit dem Benutzer über nicht-funktionale Anforderungen zu sprechen. Das beinhaltet einen Dialog zum Erfassen von NFRs, in welchem der Benutzer dem Bot kundenprojektspezifische NFRs mitteilen kann. Durch Rückfragen, Hilfestellungen und Informationen versucht der ArchBot, zusammen mit dem Benutzer, die Qualität der NFR-Spezifikationen zu verbessern.

Ebenfalls bietet der ArchBot dem Benutzer die Möglichkeit, die erfassten NFRs anzuzeigen und zu editieren.

Pattern Der ArchBot besitzt die Fähigkeit, mit dem Benutzer über Patterns zu sprechen. Der Benutzer wählt sich ein Pattern aus einer vordefinierten Liste aus, welches er in seinem Projekt verwendet. Der ArchBot liefert über entsprechende Dialoge Informationen zum ausgewählten Pattern und fragt den Benutzer bezüglich der Anwendung, Implementation und Erfahrung im Bezug zu diesem Pattern aus.

Beide Themenbereiche starten mit einer Nachfrage des ArchBot an den Benutzer, ob er mit der Thematik vertraut ist. Benötigt der Benutzer mehr Informationen zum Thema, startet der ArchBot einen entsprechenden Dialog und liefert dem Benutzer mehr Informationen und einen Link auf den passenden Wikipedia-Artikel, auf den der Benutzer klicken kann.

Feedback und Error-Handling

Wenn möglich, haben wir die Antworten des ArchBots so definiert, dass sie Bezug auf die Nachrichten des Benutzers nehmen, um den Dialog lebendig zu halten. Sollte der ArchBot den Benutzer nicht verstehen, teilt er dies dem Benutzer klar mit. Als Beispiel: Sendet der Bot eine Frage mit einer Liste mit möglichen Antworten und der Benutzer antwortet mit einer, dem Bot nicht verständlichen Nachricht, antwortet der Bot mit: «I didnt understand. Please choose an option from the list.»

Unterstützung

Wir haben verschiedene Hilfestellungen im ArchBot implementiert, die dem Benutzer zur Verfügung stehen. Der Benutzer kann jederzeit in einer Nachricht an den Benutzer um Hilfe bitten. Abhängig davon, wo der Benutzer gerade ist, haben wir unterschiedliche Hilfen implementiert. Der Bot antwortet typischerweise mit der spezifischen, im Dialog integrierten Hilfe. Ist in diesem Dialog keine Hilfe integriert, antwortet er mit der allgemeinen Hilfe.

Gesprächsabschluss

Der Benutzer kann jederzeit in einer Nachricht an den Benutzer das Gespräch beenden. Allerdings fragt der ArchBot den Benutzer in diesem Moment noch nach dessen E-Mail-Adresse, um dem Softwarearchitekten den Kontakt zum Benutzer zu ermöglichen. Anschliessend verabschiedet sich der ArchBot, speichert die Konversation in der Datenbank und beendet die Session.

5.3. Datenbank

Zum Persistieren der mit dem ArchBot geführten Konversationen, besitzt der ArchBot eine Datenbank (MongoDB). Der Administrator braucht Zugriff auf diese Daten, um eine Auswertung zu ermöglichen. Um CRUD-Vorgänge zu erleichtern, wurde die Library «Mongoose» verwendet.

5.3.1. Relevante Daten für die Auswertung

Der ArchBot speichert folgende für die Auswertung relevanten Daten ab:

- Der Nachrichtenverlauf, beziehungsweise Fragen und Aussagen des Chatbots mit entsprechenden Reaktionen des Benutzers, um den Konversationsablauf nachzuvollziehen.
- Die persönlichen Daten des Benutzers, wie den Namen und die E-Mail-Adresse, um den Benutzer zu kontaktieren.
- Die erfassten NFRs und Patterns des Benutzers, um den Einblick in das Projekt des Benutzers zu erhalten.

Ebenfalls speichert der ArchBot den Start- und Endzeitpunkt der Konversation ab, um eine Auswertung der Konversationsdauer durchzuführen.

5.3.2. Ablauf der Erfassung und Speicherung von Daten

Wie bereits im Unterabschnitt 5.2.1 beschrieben, gibt das Microsoft-Bot-Framework vor, dass während des Chatverlaufs, der ArchBot alle Daten der Konversationen in einem Objekt namens «session» speichert. Dieses «session-Objekt» besitzt einen Key «dialogData», in dem Daten für eine

Session persistiert werden, so lange diese aktiv ist,. Alle erfassten Daten der Konversation, die für die Auswertung relevant sind, speichert der ArchBot unter diesem Key. Das Objekt «dialogData» richtet sich nach den zu erfassenden Daten und wird vom ArchBot so aufgebaut, dass es direkt in der Datenbank abgelegt werden kann. Beendet der Benutzer die Konversation, oder ist er zu lange inaktiv, wird das Objekt «dialogData» in die Datenbank geschrieben.

Das Schema, nach welchem sich die Datenbank und das Objekt «dialogData» richtet, ist mit Kommentaren im Anhang C zu finden.

5.4. AdminWebServer

Zur Auswertung der Konversationen wurde ein Webserver implementiert. Der AdminWebServer stellt dem Administrator über verschiedene Endpoints, die aus den Konversationen gewonnenen Daten zu Verfügung, ohne dass er selbst die Datenbank abfragen muss. Die Implementierung des «AdminWebServer» basiert auf dem Web-Application-Framework Express.js.

Der Administrator kann im Gespräch mit dem ArchBot durch das Keyword «admin» den Administratordialog aufrufen (siehe Abbildung 5.2). Der Administrator kann aber auch direkt durch Eingabe der entsprechenden URL im Webbrowser auf die Admin Webseite zugreifen.

Im Administratordialog stellt der Bot dem Administrator Links zur Verfügung, um auf die Webseite des «AdminWebServer» zu gelangen.

Wir müssen anmerken, dass wir keine Authentifizierung umgesetzt haben, welche einen normalen Benutzer davon abhalten würde, den Administratordialog aufzurufen, beziehungsweise auf die Admin Webseite zu gelangen.

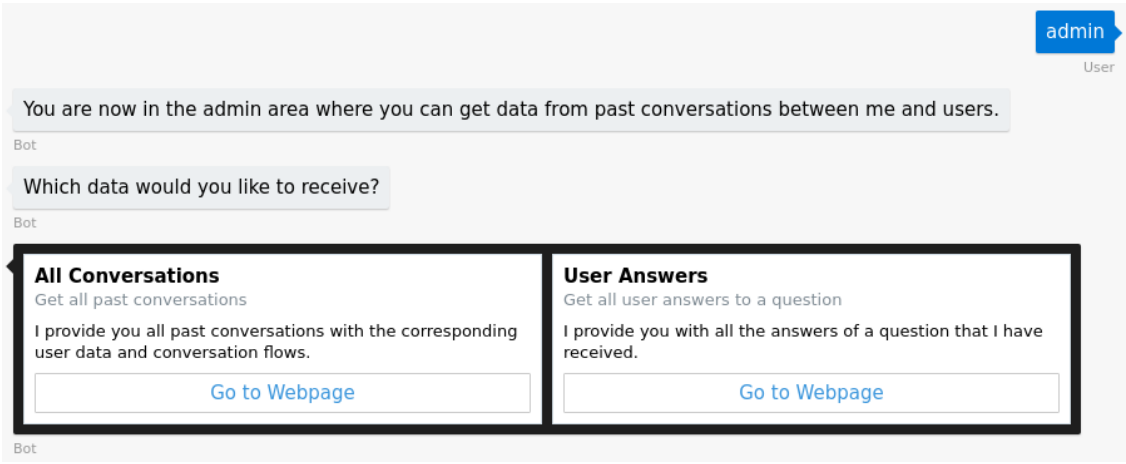


Abbildung 5.2.: Screenshot des Administratordialogs

Der «AdminWebServer» stellt dem Administrator verschiedene Ansichten und Files zur Verfügung, um die vergangenen Konversationen zu analysieren. Wie in Abbildung 5.3 zu sehen ist, kann der Administrator alle Konversationen einsehen. Durch die Buttons «Download all as JSON» und

«Download Conversation» können Details aller aufgezeichneten Konversationen, sowie einer einzigen Konversation als JSON-File heruntergeladen werden. Diese JSON-Files enthalten alle in der Datenbank gespeicherten Informationen und ermöglichen somit, den Chatverlauf nachzuvollziehen. Ebenfalls besitzt die Konversationsübersicht einen Button «Show Details», um sich die Details der Konversationen anzusehen.

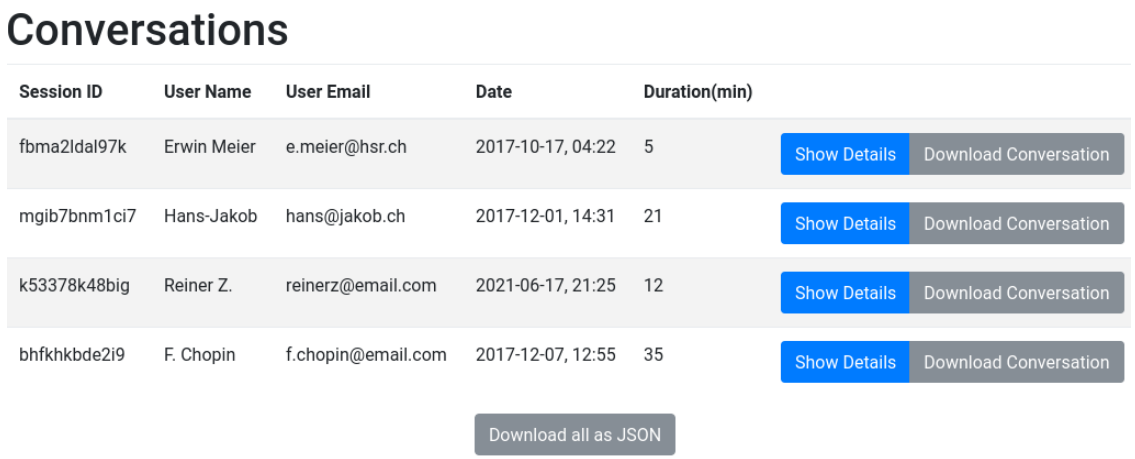


Abbildung 5.3.: Screenshot der Administratorwebseite mit Überblick über die aufgezeichneten Konversationen

Auf der «Conversation Details»-Ansicht werden zusätzliche Informationen zu den Konversationen, wie die erfassten NFRs und Patterns angezeigt (siehe Abbildung 5.4).

Auch in dieser Ansicht kann der Administrator, die Daten über die Buttons «Download NFRs» und «Download Patterns» als JSON-Files herunterladen.

Der Administrator kann, ebenfalls über den Administratordialog des Chatbots oder über die direkte URL, die Ansicht «Bot messages with user Answers» aufrufen. Wie in Abbildung 5.5 zu sehen ist, bietet diese Ansicht eine Übersicht über alle vom ArchBot gestellten Fragen. Der Button «Show» ermöglicht dem Administrator das Anzeigen aller Benutzerantworten.

5.4.1. «AdminWebServer»-Endpoints

Der «AdminWebServer» besitzt verschiedene Endpoints, die angesprochen werden können, um Webseiten oder JSON-Files zu laden.

Die verfügbaren Endpoints sind folgende:

/conversations liefert eine Ansicht mit einer Übersicht über alle Konversationen.

/conversations/download liefert ein JSON-File mit den Informationen zu allen Konversationen.

/conversations/data/:sessionID liefert eine Ansicht mit Details der mitgegebenen SessionID.

/conversations/download/:sessionID liefert ein JSON-File mit den Details der mitgegebenen SessionID.

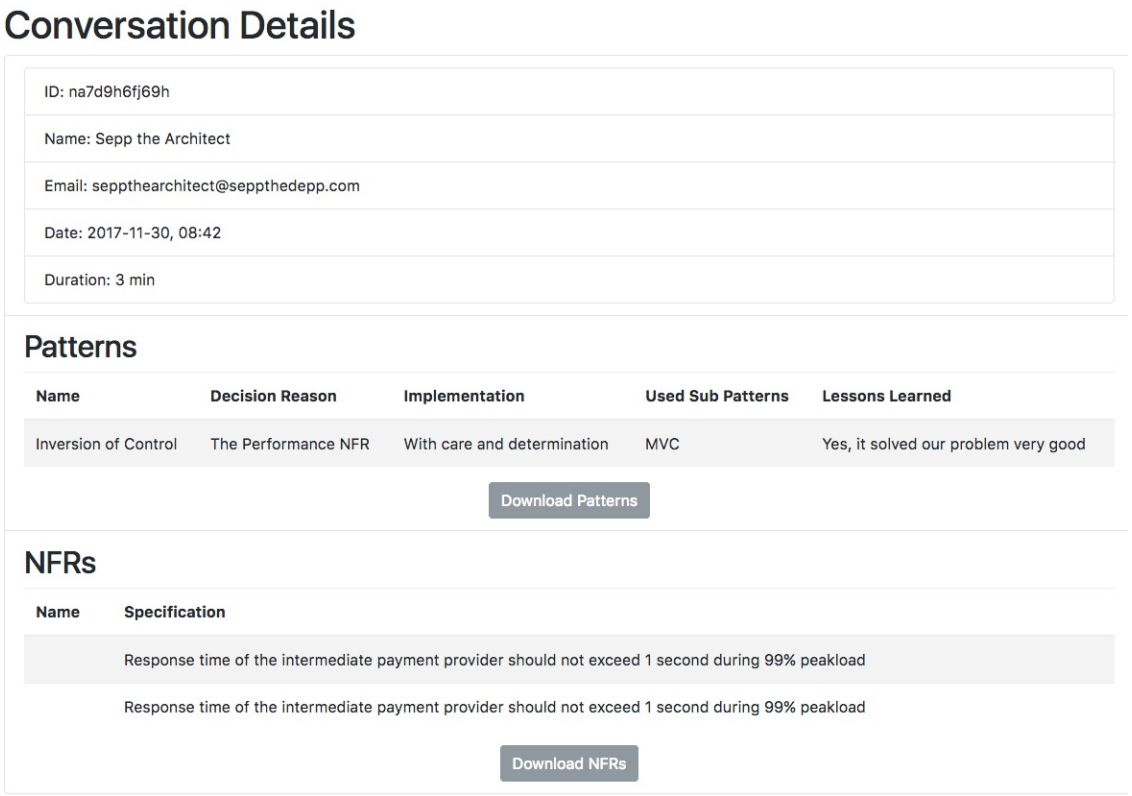


Abbildung 5.4.: Screenshot der Administratorwebseite mit Details einer Konversation

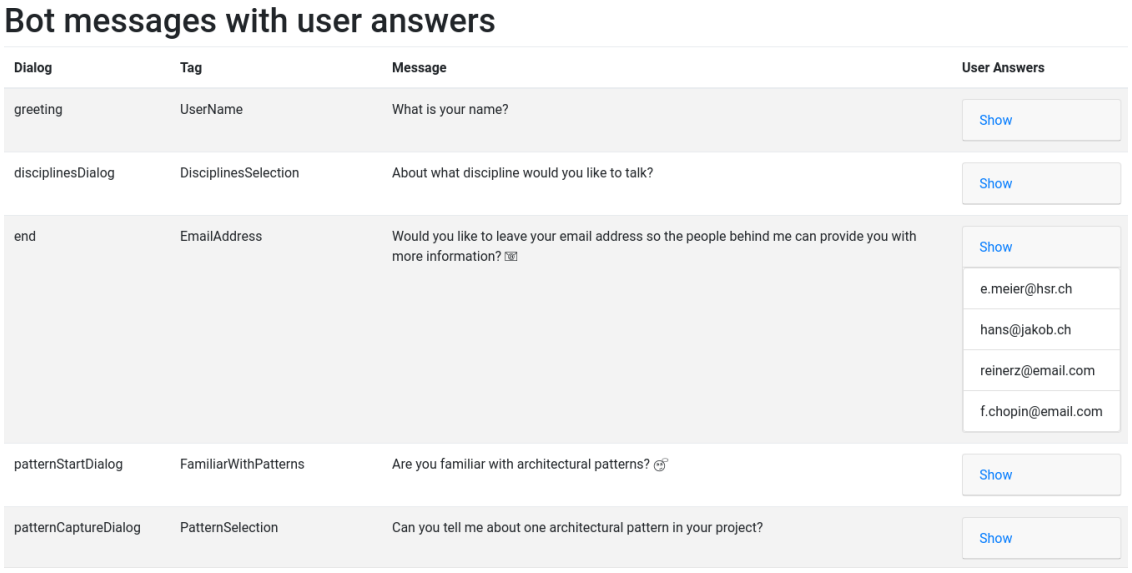


Abbildung 5.5.: Screenshot der Administratorwebseite mit Fragen des ArchBots und den dazugehörigen Antworten der Benutzer

/conversations/download/nfr/:sessionID liefert ein JSON-File mit den erfassten NFRs der mitgegebenen SessionID.

/conversations/download/patterns/:sessionID liefert ein JSON-File mit den erfassten Patterns der mitgegebenen SessionID.

/questions liefert eine Ansicht mit Fragen des ArchBots und allen dazugehörigen Benutzerantworten.

5.5. NLP-Service

Als NLP-Service verwenden wir LUIS. Die Aufgabe von LUIS ist es, die Nachrichten der Benutzer zu parsen. Zur Konfiguration von LUIS wird auf der Webseite des Dienstes eine LUIS-App registriert. Zuerst haben wir Intents für alle unsere Dialoge festgelegt und in der LUIS-App eingetragen. Anschliessend haben wir zu jedem Intent Beispielsätze hinterlegt, um den NLP-Service zu trainieren. Ein Beispiel für einen Intent mit Beispielsätzen ist in der Abbildung 5.6 zu sehen. Um ein zufriedenstellendes Ergebnis im Hinblick auf die Erkennungsrate und die Confidence zu erzielen, haben wir jeden Intent mit mindestens zehn Sätzen trainiert. Die «Bot-Middleware» des «BotServer» wurde entsprechend konfiguriert, um alle Nachrichten vom Benutzer an LUIS zu senden. LUIS sendet, wie im Unterabschnitt 4.1.2 beschrieben, die erkannten Intents und Entities an den «BotServer» zurück, dieser speichert die Antworten in das «Session-Objekt» und verarbeitet sie.

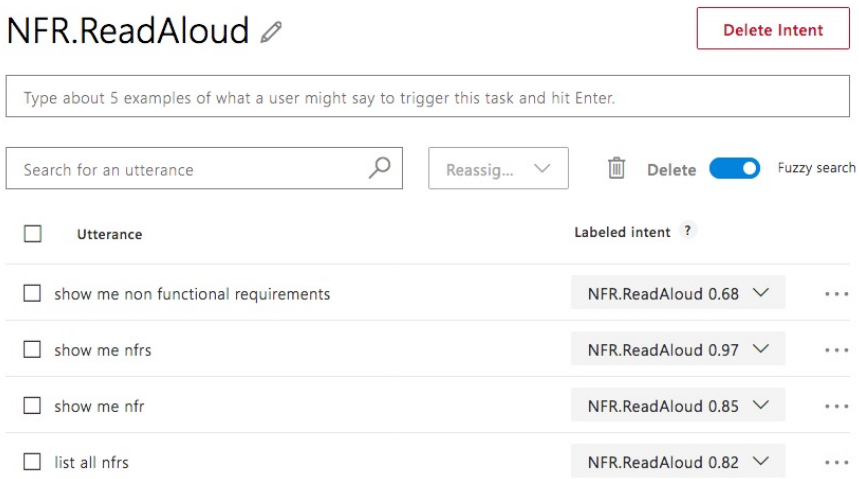


Abbildung 5.6.: Screenshot von LUIS.ai des Intent «NFR.ReadAloud» mit Beispielsätzen für das Training

5.6. Testen der Umsetztung

5.6.1. Unit-Tests

Mit jedem Commit auf das Remote Repository durchlief unser Bot einige Unit-Tests.

```
describe('ArchBot', function () {
  let bot;

  beforeEach(() => {
    bot = theBot.buildBot(connector);
  });

  it('can handle greeting and initialization', () => {
    const botTester = new _botTester.BotTester(bot)
      .sendMessageToBot('Start', 'Wie heisst du?')
      .sendMessageToBot('Sepp', 'Freut mich, Sepp!')
    return botTester.runTest();
  });
```

Listing 5.2: Testbeispiel eines kurzen Dialogs

Das in Listing 5.2 zu sehende Beispiel baut auf dem vorherig beschriebenen Beispieldialog auf. Bei der «sendMessageToBot»-Funktion wird im ersten Parameter festgelegt, was an den ArchBot gesendet wird (MessageU2B) und beim zweiten Parameter, was vom Bot als Antwort erwartet wird (MessageB2U). Unit-Tests waren für uns sehr hilfreich, da wir sonst durch den ganzen Ablauf klicken mussten, bis wir an der Stelle waren, die wir testen wollten. Leider waren wir durch das Test-Framework stark eingeschränkt, NLP-basierte Trigger funktionierten nicht, und sobald sich der Bot mit einem UI-Element zurückmeldete, waren wir nicht in der Lage, diese zu testen.

5.6.2. System-Tests

Um die Einschränkungen des «Bot Tester for Bot Builder Framework» zu umgehen, setzten wir auf Systemtests. Nach jeder grösseren Veränderung im Dialogablauf haben wir den Bot manuell getestet. Die meisten Fehler konnten so relativ schnell entdeckt und beseitigt werden.

5.6.3. Usability-Tests

Wir haben im Verlauf des Projekts Hallway-Usability-Tests durchgeführt. Dafür wurden Personen in der Nähe gefragt, ob sie den Bot ausprobieren wollen. Wir haben den Testpersonen kein Szenario vorgegeben, da der Bot möglichst intuitiv zu bedienen sein soll und er den Benutzer leiten soll. Das Feedback von diesen Tests und auch die Beobachtungen, die wir dabei gemacht haben, waren sehr hilfreich.

6. Reflexion

Dieses Kapitel bietet eine Übersicht über die Ergebnisse, die wir in dieser Arbeit erreichen konnten. Ebenso fassen wir in diesem Kapitel unsere gewonnenen Erkenntnisse zusammen und wagen einen Rückblick auf den Projektverlauf und einen Ausblick in die Zukunft.

6.1. Ergebnisse

6.1.1. Vorstudie

Ein zentraler Teil des Ergebnis dieser Studien- bzw. Bachelorarbeit ist die Vorstudie. Die Vorstudie, für welche wir knapp einen Drittel der Zeit verwendet haben, war für uns sehr hilfreich. Die meisten Quellen waren Artikel in Blogs und es war schwierig, entsprechende wissenschaftliche Quellen zu finden. Diese Feststellung bestätigt unsere Vermutung, dass Chatbots zwar seit einiger Zeit sehr gefragt sind und eine breite Verwendung gefunden haben, jedoch noch kein Standard etabliert hat. Wir haben in unserer Vorstudie zahlreiche Aspekte von Chatbots untersucht, aber auch die Grundlagen von Konversationen beleuchtet. Mit der Vorstudie ist eine Anleitung entstanden, die für Chatbot-Entwickler wertvolle Informationen beinhaltet. In der Vorstudie inbegriffen war auch das Entwickeln zweier primitiver Prototypen, um anschliessend auf dem mit dem besseren Framework aufzubauen und den ArchBot umzusetzen.

6.1.2. ArchBot

Neben der Vorstudie haben wir auf Basis der Erkenntnisse aus der Vorstudie den ArchBot entwickelt. Der Chatbot basiert auf dem Microsoft-Bot-Chatbot-Framework und greift auf Microsofts NLP-Service LUIS zurück. Der Chatbot hilft Software-Architekten dabei, nicht-funktionale Anforderungen zu spezifizieren und Architekturpatterns zu reflektieren. Ziel war es, dass der ArchBot das erste Gespräch zwischen Kunde und Softwarearchitekt vereinfacht, indem dieser bereits im Voraus wichtige Informationen zum Projekt sammelt und den Kunden mit Tipps aufklären kann. Unser Chatbot ist in der Lage, die Absichten des Benutzers zu erkennen und den passenden Dialog zu starten. Wenn der Chatbot gestartet wird, wird der Benutzer begrüsst und nach seinem Namen gefragt. Anschliessend kann der Benutzer wählen, über welches Thema er diskutieren will. Der Chatbot verfügt über zwei Hauptitents. Im Hauptintent «nicht-funktionale Anforderungen» kann der Benutzer NFRs erfassen, ansehen und löschen. Wenn der Benutzer einen neuen NFR erfasst, hakt der Chatbot nach, ob die Definition spezifisch und messbar ist. Wenn das nicht zutrifft, kann der Benutzer die Definition überarbeiten. Im Hauptintent «Architekturpatterns» fragt der Chatbot den Benutzer über seine Erfahrungen mit der Verwendung eines bestimmten Patterns ab und liefert ähnliche Patterns zum Vergleich. Zusätzlich zeigt der Chatbot dem Benutzer in beiden Hauptintents an, wo dieser Zusatzinformationen zum aktuellen Thema findet. Wenn der Benutzer die Konversation beendet, wird dieser nach seiner E-Mail-Adresse gefragt, damit ein Softwarearchitekt mit ihm Kontakt aufnehmen kann.

6.2. Erkenntnisse

6.2.1. Softwarearchitektur als Wicked Problem

Wir haben im Verlauf der Arbeit die These aufgestellt, dass das Treffen von Architekturentscheidungen vermutlich ein «Wicked Problem» ist. Ein «Wicked Problem» ist ein Problem, welches schwierig zu lösen oder sogar unlösbar ist, weil die Anforderungen unvollständig und widersprüchlich sind oder ständig verändert werden. Die Definition von «Wicked Problems» wurde 1973 erstmals von Horst Rittel und Melvin M. Webber, im Bezug auf Social Engineering, beschrieben. Sie haben dafür zehn Punkte festgelegt, die ein «Wicked Problem» definieren[12].

Zu unserer These, dass das Treffen von Softwarearchitekturentscheidungen ein «Wicked Problem» ist, haben wir eine interessante Quelle[13] gefunden, der wir im folgenden Kapitel Beachtung schenken wollen. Bei genauerer Betrachtung kann man die von Horst Rittel und Melvin M. Webber beschriebenen Punkte in drei Kategorien einteilen, da sie derselben Ursache entspringen.

Formulierung und Interpretation In diese Kategorie fallen die Punkte 1: *Es gibt keine definitive Formulierung eines «Wicked Problems»* und 9: *Das Problem kann auf verschiedene Arten beschrieben werden. Die Art der Definition bestimmt die Lösung des Problems.*

Softwarearchitekten können Architekturprobleme auf verschiedene Arten interpretieren. Die Art, wie das Problem beschrieben wird, bestimmt im Wesentlichen, wie dieses gelöst wird. Um Unklarheiten auszuschliessen, werden deshalb Formalismen angewendet. Damit stosst man jedoch wieder auf ein neues Problem. Auf der einen Seite leiten diese Formalismen den Architekten dazu an, in einer bestimmten Art und Weise über das Problem nachzudenken, auf der anderen Seite jedoch, verleiten diese Formalismen dazu, bestimmte Aspekte des Problems zu vernachlässigen[13].

Das bedeutet für unseren Chatbot, dass er in der Lage sein muss, das Problem in seiner Gesamtheit zu verstehen, um eine gute Lösung vorzuschlagen. Da jedoch Sprünge zwischen den einzelnen Dialogen schwierig zu realisieren sind, müsste sich der Chatbot am Anfang der Problembeschreibung für einen passenden Dialog entscheiden, der eine dieser Formalismen abdeckt. Durch die fixe Dialogstruktur ist es sehr wahrscheinlich, dass nicht alle Aspekte berücksichtigt werden können.

Vervollständigung und Korrektheit Zu dieser Kategorie kann man die Punkte 2: *«Wicked Problem» haben keine «Stopping Rule»*, 3: *Lösungen für «Wicked Problems» sind nicht richtig oder falsch, sondern es gibt mehrere Lösungen die besser oder schlechter sind*, 4: *Es gibt keine endgültige Überprüfungsmöglichkeit, für die Lösung eines Wicked Problems*, 6: *«Wicked Problems» haben eine unzählbare Menge von potentiellen Lösungen* und schlussendlich Punkt 8: *Jedes «Wicked Problem» kann als Symptom eines anderen Problems betrachtet werden* zählen.

Das Entwickeln von Software wird von vielen Faktoren bestimmt, die sich gegenseitig beeinflussen. Es gibt meist mehrere Stakeholder, die sehr unterschiedliche Ziele haben. Aus diesem Grund können sich Anforderungen häufig ändern. Ein Softwarearchitekt kann mit dem selben Aufwand (Personalkosten und Materialkosten) verschiedene Tools (Entwicklungsumgebungen, Beschreibungstechniken, Softwareparadigmen) verwenden, die zu sehr unterschiedlichen Lösungen führen[13].

Der Chatbot müsste damit umgehen können, dass Anforderungen ständig ändern. Wenn ein bestimmter Faktor angepasst wird, kann das Auswirkungen auf andere Faktoren haben. Das bedeutet, dass der Chatbot sehr dynamisch auf die Nachrichten des Benutzers reagieren können muss. Er muss bei jeder Nachricht vom Benutzer prüfen, ob diese Auswirkungen auf die zuvor getroffenen Entscheidungen hat um das Gespräch gegebenenfalls zu jedem Zeitpunkt in eine andere Richtung lenken können. Zudem müsste der Chatbot viel Hintergrundwissen zu den verfügbaren Tools besitzen und deren Zusammenhang zwischen der Lösung und den Kosten verstehen können.

Einzigartigkeit In diese Kategorie fallen die Punkte 5: *Jede Lösung eines Problems ist ein einmaliger Vorgang mit nur einer Chance*, 7: *Jedes «Wicked Problem» ist grundsätzlich einzigartig* und Punkt 10: *Der Planer darf nicht falsch liegen.*

Obwohl das Erkennen von Besonderheiten eines Projekts zu den Aufgaben eines Architekten gehört, können durch die Einzigartigkeit der Projekte entscheidende Faktoren lange versteckt bleiben. Werden bereits früh im Projekt Formalismen eingesetzt, steigert das die Wahrscheinlichkeit sogar noch stärker, da dabei oft Ähnlichkeiten zu vergangenen Projekten gesucht werden und dadurch die Sichtweite noch stärker eingeschränkt wird. Da Softwareprojekte oft ein begrenztes Zeit- und finanzielles Budget haben, sind die Moglichkeiten verschiedene Losungen auszuprobieren begrenzt. Die Lösung muss also zu einem akzeptablen Ergebnis führen[13].

Der verwendete NLP-Service LUIS, ist wie grundsätzlich alle NLP-Services, darauf trainiert, Ähnlichkeiten zu erkennen. Ähnlichkeiten bei Softwareprojekten sind relativ einfach zu identifizieren. Die Herausforderung ist jedoch, die Besonderheiten zu erkennen, da diese die grössten Auswirkungen auf den individuellen Lösungsvorschlag haben. Für den Chatbot wird es deshalb schwierig, zu entscheiden, ob der Lösungsvorschlag gut genug ausgearbeitet ist und er alle Besonderheiten berücksichtigt hat.

Zusammenfassung

Wir schliessen aus diesen Punkten, dass das Treffen von Architekturentscheidungen tatsächlich ein «Wicked Problem» ist. Softwarearchitektur ist äusserst komplex und wird von zahlreichen Faktoren beeinflusst, über die der Architekt selber keine Entscheidungsmacht hat. Um ein guter Softwarearchitekt zu sein, ist Erfahrung in Kombination mit einer intuitiven Wahrnehmung unerlässlich, damit erfolgreich zwischen den einzelnen Spannungsfelder vermittelt werden kann. Der Beruf eines Softwarearchitekten dürfte deshalb in naher Zukunft nicht von einem Chatbot übernommen werden. Trotzdem kann ein Chatbot Abhilfe schaffen und dabei helfen, Informationen zu sammeln und einen Rahmen für das Gespräch vorzugeben.

6.3. Rückblick

Wir sind der Meinung, dass wir uns für eine gute Herangehensweise zum Bewältigen der Aufgabe entschieden haben. Die Vorstudie half uns beim Einarbeiten in die Thematik und beim anschliessenden Entwickeln des Bots. Wir konnten die funktionalen Anforderungen (Informationssammlung, Auslesen der Konversationen) vollumfänglich umsetzen. Bis auf zwei Ausnahmen (Channel Integration, Erkennen der Absicht des Benutzers), die nicht getestet wurden, haben wir auch alle

nicht-funktionalen Anforderungen erfolgreich umgesetzt. Die grösste Herausforderung war das Festlegen des Dialogablaufs. Sprünge zwischen Dialogen waren schwierig zu realisieren und es war auch schwierig, dem ArchBot einen Charakter zu verpassen und menschlich wirken zu lassen. Wir haben die Komplexität unterschätzt und mussten uns deshalb nur auf zwei verschiedene Themenbereiche beschränken, die der Chatbot abdecken kann. Auch die Auswahl des JavaScript SDK bereitete uns anfangs Mühe, da wir beide noch nicht auf viel Erfahrung zurückgreifen konnten. Weitere Schwierigkeiten bereitete das Testing Framework, welches nicht allen unseren Ansprüchen genügte. Wir verloren viel Zeit durch manuelles Testen, weil wir immer durch den Dialog klicken mussten, bis wir zur geänderten Stelle gelangten und diese testen konnten. Beim nächsten Mal würden wir bei der Evaluation der verschiedenen Frameworks mehr darauf achten, wie gut getestet werden kann. Ein weiteres Problem war, dass die Anzahl kostenloser Anfragen auf den NLP-Service LUIS beschränkt waren. Diese Beschränkung war uns bereits während der Vorstudie bekannt, dachten allerdings, dass 10'000 API-Calls reichen würden. Weil aber jede Nachricht des Benutzers an LUIS gesendet wird, war das nicht der Fall. Um diese Begrenzung zu umgehen, haben wir einen zweiten Account bei LUIS eingerichtet.

6.4. Ausblick

In Zukunft könnte man unseren Chatbot von retrieval-based auf generative-based umbauen. Der Chatbot könnte seine Antworten selber generieren und mit aufgezeichneten Gesprächen zwischen Softwarearchitekt und Kunde trainiert werden. Zu erwartende Fortschritte im Bereich Machine Learning und Natural Language Processing lassen darauf hoffen, dass das Entwickeln von generative-based Chatbots einfacher wird. Bei unserem Chatbot handelt es sich um einen einfachen Prototypen. Auf die Adminfunktion kann jeder Benutzer zugreifen, der weiss, wie man den entsprechenden Dialog auslöst oder die URL zur Webseite kennt. Sollte man diesen Chatbot produktiv einsetzen wollen, müsste diese Sicherheitslücke beseitigt werden, um die Daten zu schützen. Wir konnten aus Zeitgründen auch nicht alle in der Vorstudie beschriebenen Best-Practices, wie zum Beispiel das Einbauen von häufigen Fragen und typischen Gesprächswörtern, umsetzen. Zum jetzigen Zeitpunkt läuft der Chatbot nur im Microsoft-Bot-Emulator. Der Bot kann jedoch ohne grosse Anpassungen auf anderen Channels, wie zum Beispiel Slack oder Telegram bereit gestellt werden. Vorstellbar ist auch die Integration von mehr Content, um neben Architekturpatterns und nicht-funktionalen Anforderungen weitere Themengebiete abzudecken und somit ein für Architekten wertvolles Werkzeug zu bieten.

Abbildungsverzeichnis

Literatur

[1] 1253139114830116. *Chatbot User Experience 101: Vital Tips to Improve Chatbot UX*. Aufgerufen am 16.10.2017. 2017-Oktober. URL: https://uxdesign.cc/chatbot-user-experience-101-vital-tips-to-improve-chatbot-ux-ca02bc36587c.

[2] *Actions and Parameters | Dialogflow*. Aufgerufen am 17.10.2017. URL: https://dialogflow.com/docs/actions-and-parameters.

[3] Josh Barkin. *When Bots Fail At Conversation Being Janis Medium*. Aufgerufen am 12.10.2017. 2016-08. URL: https://medium.com/janis/when-bots-fail-at-conversation-d7419605f5cc.

[4] Paul Boutin. *Does a Bot Need Natural Language Processing? Chatbots Magazine*. Aufgerufen am 16.10.2017. 2017-04. URL: https://chatbotsmagazine.com/does-a-bot-need-natural-language-processing-c2f76ab7ef11.

[5] Denny Britz. *Deep Learning for Chatbots, Part 1 Introduction*. Aufgerufen am 05.10.2017. 2016-Mai. URL: http://www.wildml.com/2016/04/deep-learning-for-chatbots-part-1-introduction/.

[6] Alex Debecker. *5 Chatbot Questions You Must Be Ready For Alex Debecker ubisend*. Aufgerufen am 12.10.2017. 2017-04. URL: https://blog.ubisend.com/optimise-chatbots/chatbot-questions-you-must-be-ready-for.

[7] *Entities | Dialogflow*. Aufgerufen am 17.10.2017. URL: https://dialogflow.com/docs/entities.

[8] Dan P. Gailey. *Bot Theory: A concise introduction*. Aufgerufen am 16.10.2017. 2016-Dezember. URL: https://github.com/dpgailey/bot-theory-book/blob/master/BotTheory.pdf.

[9] Stefan Kojouharov. *Quora How can I build an intelligent chat bot?* Aufgerufen am 05.10.2017. 2017-01. URL: https://www.quora.com/How-can-I-build-an-intelligent-chat-bot.

[10] Jesús Martín. *Design Framework for Chatbots Chatbots Magazine*. Aufgerufen am 12.10.2017. 2017-02. URL: https://chatbotsmagazine.com/design-framework-for-chatbots-aa27060c4ea3.

[11] *NLP Intent | NLP World - Glossary*. Aufgerufen am 16.10.2017. URL: https://www.nlpworld.co.uk/nlp-glossary/i/intent/.

[12] Horst W. J. Rittel und Melvin M. Webber. "Dilemmas in a General Theory of Planning". In: *Design: Critical and Primary Sources* (). DOI: 10.5040/9781474282932.0015.

[13] Bernard Robertson-Dunn und Bernard Robertson-Dunn. "A Problem Oriented Enterprise Architecture Approach Applied to Wicked Problems". In: *Enterprise Architecture for Connected E-Government* (). DOI: 10.4018/978-1-4666-1824-4.ch002.

[14] Nikhil Savaliya. *Quora How do i develop chatbot using python from scratch?* Aufgerufen am 11.10.2017. 2017-08. URL: https://www.quora.com/How-do-I-develop-chatbot-using-python-from-scratch.

[15] Eunji Jinny Seo. *19 Best UX Practices for Building Chatbots Chatbots Magazine*. Aufgerufen am 12.10.2017. 2017-01. URL: https://chatbotsmagazine.com/19-best-practices-for-building-chatbots-3c46274501b2.

[16] Amir Shevat. *Designing bots: creating conversational experiences*. OReilly Media, 2017. Kap. What Are Bots?

[17] Kumar Shridhar. *Rule based bots vs AI bots Medium*. Aufgerufen am 04.10.2017. 2017-Mai. URL: https://medium.com/botsupply/rule-based-bots-vs-ai-bots-b60cdb786ffa.

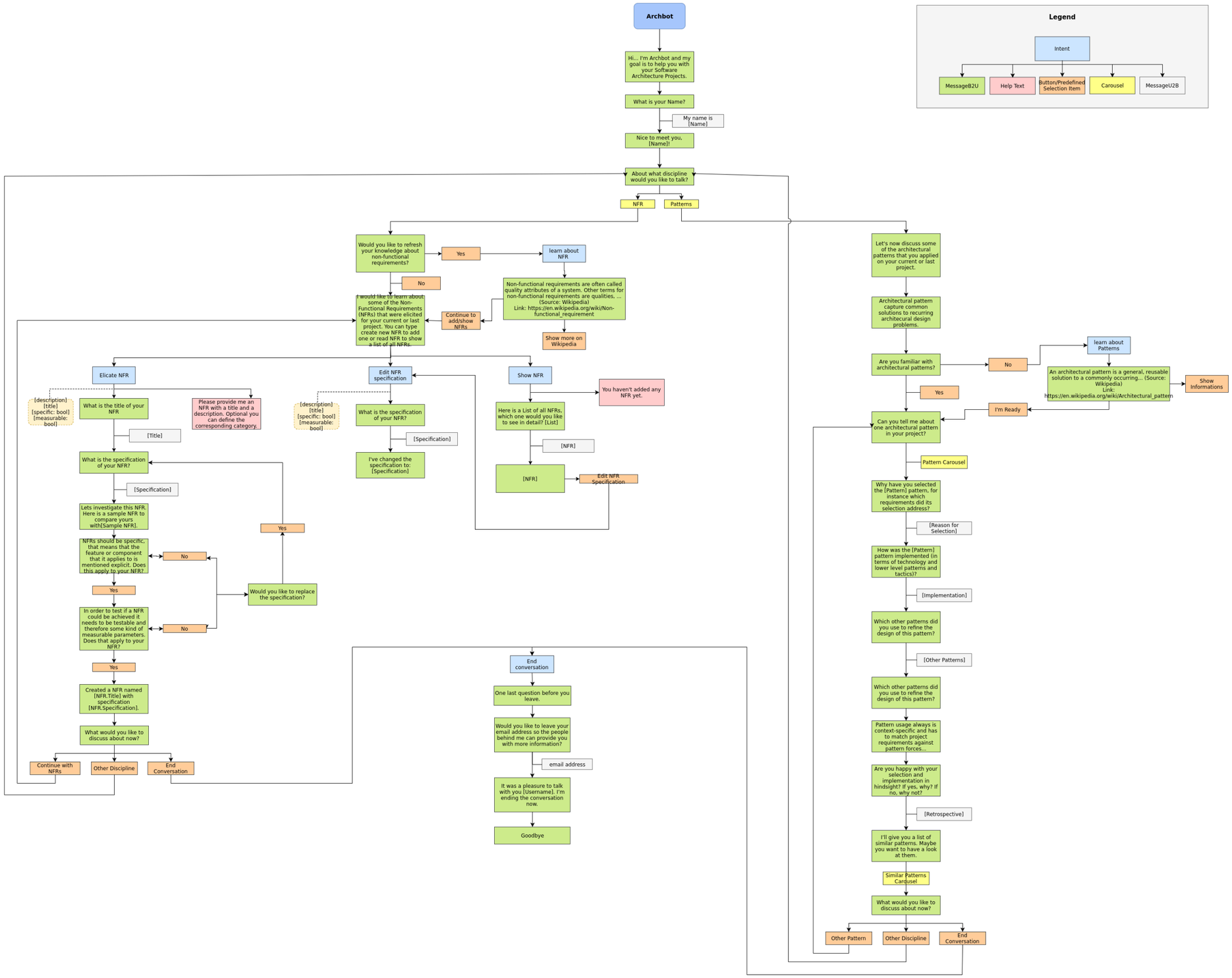
[18] Robert Standefer. *How Bot Service works - Bot Service*. Aufgerufen am 17.10.2017. URL: https://docs.microsoft.com/en-us/bot-framework/overview-how-bot-framework-works.

[19] Joe Toscano. *The ultimate guide to chatbots*. Aufgerufen am 16.10.2017. 2016-11. URL: https://www.invisionapp.com/blog/guide-to-chatbots/.

A. Nutzwertanalyse

		Nutzwertanalyse																					
Bewertungskriterium	Gewichtung in %	Frameworks																					
		Dialogflow			MS Bot Framework			wit.ai			IBM Watson			FB Messenger-Platform			Botpress.io			Botkit		
		Bew.	Nutzw.	Kommentar	Bew.	Nutzw.	Kommentar	Bew.	Nutzw.	Kommentar	Bew.	Nutzw.	Kommentar	Bew.	Nutzw.	Kommentar	Bew.	Nutzw.	Kommentar	Bew.	Nutzw.	Kommentar
Mögliche Channels	10%	100	10	FB Messanger, Slack, Web, Kik, Line, Skype, Spark, Telegram, Twitter, Viber	100	10	Bing, Cortana, FB Messanger, Kik, Slack, Skype, Web, Microsoft Teams, Email, Telegram	40	4	Nur HTTP API verfügbar	80	8	Slack, Webchat, FB Messanger	10	1	Facebook Messenger	80	8	Facebook, Skype, Microsoft Team, Skype, Slack, Whatsapp, kik	80	8	Slack, Cisco Spark, Microsoft Team, Facebook
Lizenz	20%	100	20	Apache 2.0	80	16	MIT	100	20	You are hereby granted a non-exclusive, worldwide, royalty-free license to use, copy, modify, and distribute this software in source code or binary form for use in connection with the web services and APIs provided by Wit.ai.	10	2	Closed Source und kostet ab 25 Intents	80	16	MIT	70	14	GNU Affero General Public License v3.0	80	16	MIT
Mächtigkeit der NLP und AI Engine	10%	80	8	Rule-based mit Machine Learning. Kann als Middleware Funktionieren	100	10	Verwendet LUIS (Language Understanding Intelligence Service), Bietet Text Analyitics API um Sentiment, Key Phrases, Topics und Sprache auszulesen, Bing Spell Check API um Rechtschreibung zu prüfen und Erkennen von Namen, Markennamen und Dialekt, Linguistic Analysis API, Machinebased QnA Maker	80	8	Besitzt eingeschränktes NLP(suche von Vorname nicht möglich), ermöglicht erstellen von Regeln und besitzt SDKs für verschiedene Platformen. Kann als Middleware Funktionieren	100	10	Sehr Mächtig, basiert auf Neuronale Netzwerke die mit Wikipedia Artikel trainiert wurden	70	7	Unterstützt NLP nur in Englisch	100	10	Viele Möglichkeiten durch ein "highly modular ecosystem". Ist Modular aufegbaut und kann um benötigte Komponenten erweitert werden. Kann Beispielsweise "Dialogflow" als midleware verwenden für NLP.	40	4	Kein integriertes NLP sondern von Haus aus nur Regex based. Es kann LUIS von Microsoft als NLP verwendet werden.
Importmöglichkeiten des Q&A-Content	10%	80	8	JSON	80	8	JSON	80	8	JSON	80	8	CSV, JSON	10	1	Nichts gefunden	100	10	Abhängig von Modul, CMM (Markdown)	80	8	JSON
Zukunftssicherheit	15%	90	13,5	Breite Verwendung (unter anderem Wall Street Journal, Mercedes, KLM). Grosses Entwicklerteam, Aktives Repository	90	13,5	> 7000 Stars auf Github, ist jedoch Preview (Beta), Bekanntes Unternehmen	90	13,5	Allgemein Aktiv. Kurzlich wurden grössere Änderungen vorgenommen welche Migration von Bots voraussetzte. Gewisse Sicherheit durch grösse des Unternehmens (Facebook)	90	13,5	Sehr hoch, in IBM Watson wird viel investiert, mit IBM auch eine mächtiger und traditionsreicher Hersteller	90	13,5	Hoch, wird von Facebook entwickelt, Facebook investiert viel in den FB Messenger	90	13,5	Werben mit Zukunftssicherheit. Aktiv auf Github	100	15	> 7000 Stars auf Github, seit zwei Jahren aktiv
Natural Language Support	5%	100	5	Deutsch, Spanisch, Französisch, Englisch, Russisch, Japanisch, Chinesisch	100	5	Unterstützt verschiedene Sprachen (Chinesisch, Englisch, Deutsch, Spanisch,...)	100	5	English, Deutsch, Französisch, Italienisch, uvm.	40	2	Englisch, Japanisch	80	4	Wird unterstützt	80	4	Abhängig von eingebunden Modulen	40	2	Regex => alle, NLP abhängig von externem NLP
Programmiersprache	20%	100	20	SDKs für JS, Java, Node. js, .NET, HTML, und weitere	90	18	.Net, Node.js, REST	40	8	SDKs für Node.js sowie Libraries für Ruby und Python und eine HTTP-API	90	18	Node SDK, Java SDK, Python SDK, iOS SDK, Unity SDK	10	2	Nichts gefunden	70	14	JavaScript	70	14	Framework ist in JavaScript geschrieben. Bot kann mittels Scripts auf der Webseite konfiguriert werden. Besitzt API um mit dem Bot zu kommunizieren. Programmiersprachen unabhängig dank API.
UI Design Elemente	10%	80	8	Channelspezifisch, unterstützt Text, Bilder und Cards (Beinhaltet Bild, Titel Untertitel und Buttons)	80	8	Channelspezifisch, WebChat bietet (Cards, Attatchments)	80	8	Channelspezifisch, unterstützt Text, Bilder und Cards (Beinhaltet Bild, Titel Untertitel und Buttons)	80	8	Channelspezifisch, muss speziell definiert werden	80	8	Einfach umzusetzen, da der Messenger das Erstellen von Cards, Quick Responses vereinfacht	80	8	Sehr grosse Auswahl an UI Design Elementen	40	4	BotMenu(https://msdn.microsoft.com/en-us/microsoft-teams/botmenu)
Nutzwertsumme			**92,5**			**88,5**			**74,5**			**69,5**			**52,5**			**81,5**			**71**	

B. Conversationflow Diagramm



C. Datenbank Schema

```
{
    // Einzigartige ID die vom Archbot für jede Session vergeben wird.
    sessionID: {
        type: String
    },
    // Zeitpunkte für den Start und das Ende der Session
    date: {
        // Zeitpunkt des Sessionstarts
        startDate: {
            type: Date
        },
        // Zeitpunkt des Sessionendes
        endDate: {
            type: Date
        }
    },
    // Benutzerdaten der Session
    user: {
        // ID die von Archbot für den Benutzer vergeben wird.
        userID: {
            type: Number
        },
        // Namen des Benutzers
        name: {
            type: String
        },
        // E-Mail-Adresse des Benutzers
        email: {
            type: String
        }
    },
    // Vom Benutzer aufgerufene Dialoge
    dialogs: [{
        // Name des aufgerufenen Dialogs
        name: {
            type: String
        },
        // Nachrichten die im Dialog gesendet wurden
        messages: [{
            // Absender der Nachricht
            sender: {
                type: String,
                enum: ['bot', 'user']
            },
            // Zeitpunkt der Nachricht
            date: {
```

```
            type: Date
        },
        // Tag der Nachricht um Aussagen von Bot und User
                zusammen zu führen
        tag: {
            type: String
        },
        // Textinhalt der Nachricht
        content: {
            type: String
        },
        // Vom NLP erkannter Intent einer User-Message
        intent: {
            type: String
        },
        // Confidence des NLP des erkannten Intents
        confidence: {
            type: Number
        }
    }],
}],
// Vom Benutzer behandelte Patterns
patterns: [{
    // Name des Pattern
    name: {
        type: String
    },
    // Grund für die Patternwahl
    reason: {
        type: String
    },
    // Umsetztung der Implementierung des Patterns
    implementation: {
        type: String
    },
    // Andere verwendete Pattern im Zusammenhang mit dem Pattern
    subPatterns: {
        type: String
    },
    // Lessons Learned bezüglich des Pattern
    lessonsLearned: {
        type: String
    }
}],
// Vom Benutzer erfasste NFRs
nfr: [{
    // Name/Titel der NFR
    title: {
        type: String
    },
```

```
        // Speizifikation der NFR
        text: {
            type: String
        }
    }]
},
{minimize: false}
```

D. Installationsanleitung

Die Installationsanleitung beinhaltet Informationen, wie der Chatbot installiert und verwendet wird.

D.1. Anforderungen

Um den Archbot zu betreiben werden folgende Applikationen benötigt:

- Node.js
- Docker Community Edition
- Bot Framework Emulator

D.2. Installation

Zur Verwendung muss Archbot installiert werden.

```
cd src
npm install
```

D.3. Verwendung

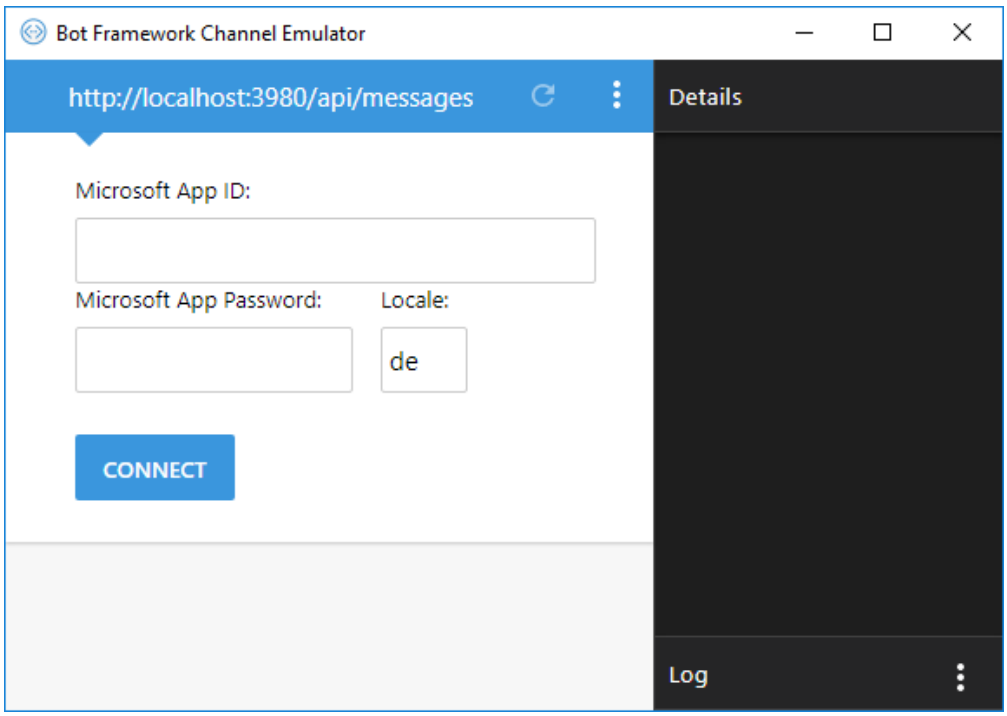
Starten der MongoDB in einem Docker-Container mit einem persistenten Verzeichnis. Dazu muss /my/own/datadir mit einem lokalen Verzeichnis ersetzt werden.

```
docker run -d -p 27017:27017 -v <db-dir>/:data/db mongo
```

Starten des Archbots

```
### Im Verzeichnis Archbot/src
npm start
```

Verbinden des Bot Framework Emulators mit dem Archbot über die Endpoint-URL: http://localhost:3980/api/r



D.4. Auslesen der Konversation

Alle Konversationen und Fragen mit den zugehörigen Antworten werden in der Datenbank gespeichert, und sind entweder über den Adminbereich des Bots, oder direkt über die URLs erreichbar.

- Der Adminbereich des Archbots ist erreichbar durch eingabe des Triggerwortes «admin ».
- Konversation: http://localhost:5000/conversations
- Fragen-Antworten: http://localhost:5000/questions

D.5. Konfiguration

Für die Weiterentwicklung oder Anpassung an die Umgebung sind hier zusätzliche Informationen zur Konfiguration beschrieben.

D.5.1. LUIS

Die Verwendung des NLP-Service LUIS setzt ein Microsoft-Account voraus. Es besteht bereits ein Account, welcher vom Archbot standardmässig verwendet wird.

E-Mail Adresse: archbot_hsr@hotmail.com

Passwort: BASA2017BASA

Im Environment File .env ist die ‘LUIS_MODEL_URL‘mit diesem Account verbunden.

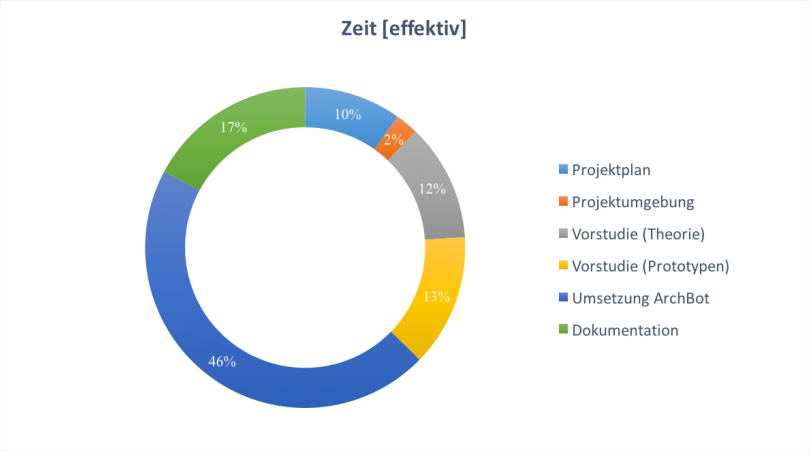
Neuer Account

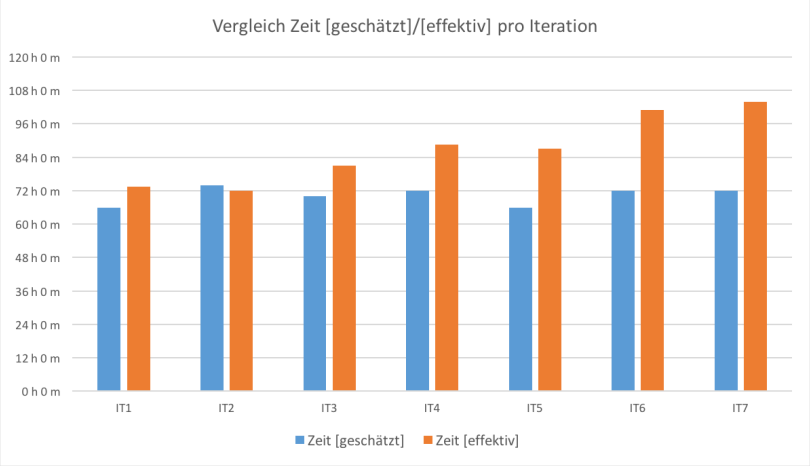
Sollte der vorbereitete Account nicht mehr erreichbar sein muss ein neuer Account bei LUIS erstellt werden. Anschliessend kann man das LUIS Configuration File (JSON) importieren.

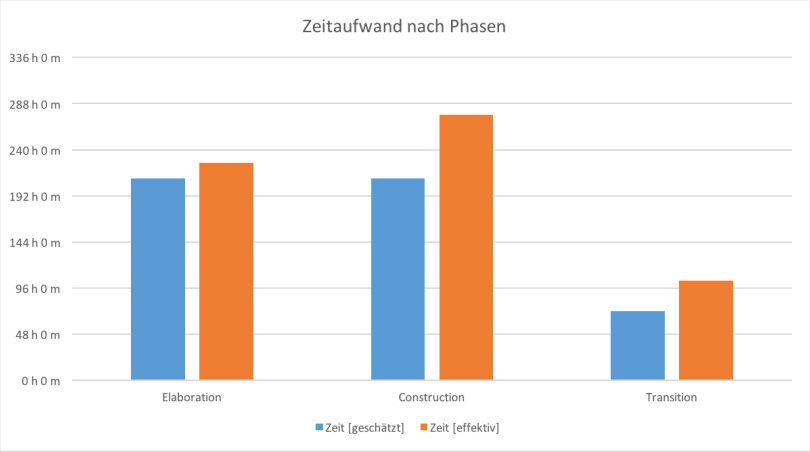
D.5.2. Ports und Verbindungen

Die verwendeten Ports des Botserver sowie des Webservers für den Adminbereich können im Environment File .env angepasst werden. Sollte der Port der MongoDB geändert werden muss die Endpoint-URL im file ‘/src/configs/config ‘angepasst werden.

E. Zeiterfassung







F. Projektplan

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

IFS INSTITUTE FOR SOFTWARE

ArchBot
architecture is my business

Projektplan

Studien-/Bachelorarbeit HS 2017: Chatbot für Software Design und Review

Abteilung Informatik
Hochschule für Technik Rapperswil

Autoren: Samuel Krieg, Ennio Meier
Betreuer: Prof. Dr. Olaf Zimmermann
Datum: 22. Dezember 2017

Inhaltsverzeichnis

1. Projektübersicht

1.1. Ziel des Projekts

Virtueller Q&A-Assistant für agile Software-Architekturarbeiten, welcher zwei grundlegende Funktionen bietet:

1. Informationssammlung im Forward Engineering, welche dazu dient, eine Offerte für ein Softwareprojekt zu erstellen.

2. Review von bestehender Software, welche als Resultat einen Software-Healthcheck empfielt.

1.2. Projektaktivitäten

Evaluation Evaluation und Auswahl eines passenden Bot Development Framework gemäss folgender Kriterien:

- Deploymentmöglichkeiten (Standalone, Web, Cloud)
- Lizenz
- Mächtigkeit der Natural Language Processing und der Artificial Intelligence (AI) Engine
- Importmöglichkeiten des Q&A Content (Formate, APIs)
- Zukunftssichere Existenz des Frameworks
- Natural Language Support (Mehrsprachigkeit)

Integration von Q&A-Content

- Business Value und Context
- User Stories
- Nichtfunktionale Anforderungen und Quality Attribute
- Patterns, Decisions
- Software-Healthcheck

Implementation Implementation eines Web-Prototyps unter Verwendung des ausgewählten Bot Development Framework. Besonderen Wert wird auf folgende Punkte gelegt:

- agiles Vorgehen
- kognitive Fähigkeit der erstellten Regeln (Einbezug der Antworten auf Folgefragen, Ergebnis/Erfolgsrate)
- AI/Search Metriken wie Precision und Recall
- Erweiterbarkeit
- Benutzerfeedback
- Externer Input/Feedback wird erwartet

1.3. Lieferumfang

Bei Abschluss des Software Engineering Projekts werden folgende Dokumente abgeliefert:

- Projektplan
- Source Code
- Gitlab Wiki
  - Vorstudie
  - Architekturdokumentation
  - Installationsanleitung
  - Benutzeranleitung
  - Technische Dokumentation
- Projektdokumentation
- Schlussberichte

Die Abgabe erfolgt in der vom Betreuer gewünschten Form (gedruckt, digital auf Datenträger).

1.4. Dokumentenplan

Der Dokumentenplan ändert sich ständig und die meisten Dokumente werden in das Gitlab Wiki integriert. Der Dokumentenplan umfasst unter anderem folgende Artefakte:

Projektplan Der Projektplan beinhaltet die grobe Planung des Projekts und legt Massnahmen fest, um eine hohe Qualität zu erzielen.

Anforderungsanalyse In der Anforderungsanalyse werden entscheidende funktionale und nichtfunktionale Anforderungen für das Projekt definiert.

Vorstudie In der Vorstudie wird analysiert, welche Faktoren für das Entwickeln eines Bots bedeutend sind und welche Bot Development Frameworks für das Projekt in Betracht zu ziehen sind. Auch werden Tests definiert, um Frameworks auf Erfüllen dieser Faktoren zu testen. Teilartefakte der Vorstudie sind:

- Prototypevaluationsszenario
- Kriterienkatalog zur Frameworkevaluation
- Definition der Testmetriken

Architekturdokumentation Beinhaltet unter anderem die logische und physische Architektur und Entwurfsentscheidungen. Ist stark von den in der Vorstudie getroffenen Entscheidungen abhängig, weshalb noch nicht klar ist, woraus sich die Architekturdokumentation genau zusammensetzt.

Benutzeranleitung Die Benutzeranleitung zeigt auf, wie der Chatbot Client installiert und benutzt werden kann.

Installationsanleitung Die Installationsanleitung beinhaltet Informationen, wie der Chatbot deployed wird und die nötigen Komponenten gestartet werden (z.B. Server)

Dokumentation Die Dokumentation bezeichnet das abzugebende Dokument und wird nicht im Gitlab Wiki erstellt, kann jedoch Teile des Wikis enthalten.

2. Managementabläufe

2.1. Vorgehensmodell zur Softwareentwicklung

Wir haben uns für ein hybrides Vorgehensmodell entschieden und kombinieren das Unified Process Modell mit den Phasen Inception, Elaboration, Construction und Transition mit agiler Arbeitsweise, in dem wir innerhalb der einzelnen Phasen agil arbeiten. So werden in den Phasen vorerst nur grobe Arbeitspakete definiert, die später auf fein granulare Issues aufgespaltet werden.

2.2. Gemeinsame Arbeitszeit

Am Montag Nachmittag und Dienstag ganztags arbeiten wir zusammen am Projekt. Die restliche Arbeitszeit ist von den Teammitgliedern selbstständig einzuteilen.

2.3. Zeitliche Planung

2.3.1. Phasen

Mittels Labels werden zusammengehörige Issues (z.B. UI Design, Architektur) gekennzeichnet. Auch die Priorität einzelner Issues wird mit Labels festgelegt (High, Medium, Low). Issues die in der selben Phase erledigt werden, werden in Gitlab als Milestones zusammengefasst. Die detaillierte Planung der Phasen findet am Ende der vorhergehenden Phase statt.

Elaboration Diese Phase umfasst die Iterationen IT1, IT2 und IT3. Es wird der Projektplan erstellt, die Projektumgebung aufgesetzt, Anforderungen definiert und die Vorstudie, welche dabei hilft ein passendes Framework zu wählen gemacht. In der letzte Phase IT3 wird die die konzeptionelle Architektur definiert. Zudem werden mit der aus der Vorstudie resultieren Auswahl von Frameworks Prototypen erstellt um eine finale Auswahl des Frameworks zu ermöglichen.

Construction Die Constructionphase beinhaltet die Iterationen IT4, IT5 und IT6 in denen die Alpha, Beta und Final Version des Chatbots umgesetzt und jeweils getestet wird.

Transition Die letzte Phase beinhaltet lediglich die Iteration IT7. In dieser Phase werden die abzugebenden Dokumente fertiggestellt, die Installationsanleitung und Benutzeranleitung geschrieben und der Sourcecode für die Abgabe überarbeitet (Refactoring).

2.3.2. Iterationen

Die Iterationsdauer beträgt zwei Wochen. So stehen pro Woche 40 Arbeitsstunden zur Verfügung. Diese setzen sich aus den vorgesehenen 16 Stunden pro Woche bei der Studienarbeit und aus den 24 Stunden pro Woche bei der Bachelorarbeit zusammen. Abgezogen werden jedoch 4 Stunden pro Woche für Meetings und und weitere 4 Stunden für die Risikoabdeckung und das Projektmanagement. Die effektive Arbeitszeit am Projekt beträgt demnach 60 Stunden pro Iteration.

IT1: Projektplan und Projektumgebung

- Projektplan erstellen
- Aufsetzen der Projektmanagementtools
- Festlegen des Workflows

IT2: Anforderungen und Analyse

- Einarbeiten in die Chatbot Thematik
- Anforderungen definieren
  - Funktionale Anforderungen durch Use Cases (Diagramm, Textuell)
  - Nichtfunktionale Anforderungen definieren und Ergänzen der Risiken
- Vorstudie
  - Erstellung eines Kriterienkatalog zur Evaluation von Bot Development Frameworks
  - Definition eines Prototypsevaluationsszenario
  - Treffen einer engeren Auswahl von Frameworks zur detaillierteren Betrachtung und Dokumentation der Entscheidungen, Vor-/Nachteile der verschiedenen Frameworks anhand des definierten Kriterienkatalog
  - Definition von Testmetriken unter anderem Usability Test, AI/Search Metriken wie Precision und Recall, kognitive Fähigkeiten der erstellten Regeln

IT3: Prototypen und Architektur

- Konzeptionelle Architektur definieren
- Erstellen von Prototypen unter Verwendung des Prototypevaluationsszenarios basierend auf den Frameworks (voraussichtlich drei), die es in die engere Auswahl geschafft haben.
- Testen der erstellten Prototypen auf Erfüllen der definierten Testmetriken und Benutzerfreundlichkeit

- Finale Entscheidung für ein Framework
- Erstellen und Abarbeiten der End of Elaboration Checkliste

IT4: Alpha

- Aufsetzen der für das Framework passender Entwicklungsumgebung. Beinhaltet auch das Aufsetzen der CI (Continuous Integration), CD (Continuous Delivery) und IDE.
- UI Design: Erstellen eines Benutzerinterface
- Entwicklung der Alphaversion aufbauend auf dem Prototyp
- Integration des Q&A Content welcher vom Betreuer geliefert wird
- Testen der Alphaversion auf auf Erfüllen der definierten Testmetriken und Benutzerfreundlichkeit

IT5: Beta

- Entwickeln der Betaversion aufbauend auf Alphaversion
- Testen der Betaversion auf Erfüllen der definierten Testmetriken und Benutzerfreundlichkeit

IT6: Final

- Entwickeln der finalen Version aufbauend auf Betaversion
- Testen der finalen Version auf Erfüllen der definierten Testmetriken und Benutzerfreundlichkeit

IT7: Transition und Abgabe

- Erstellen einer Benutzeranleitung
- Erstellen einer Installationsanleitung
- Erstellen einer Demo
- Fertigstellung der Dokumente
- Code für Abgabe vorbereiten

PH0: Metawork

- Zusätzlich zu den vorhandenen Iterationen gibt es einen Phase Metawork, welcher sich über die ganze Projektdauer erstreckt. Dieser Milestone wird dazu verwendet um wiederkehrende und andauernde Issues zu verwalten. Beispiele für diese Issues sind unter

anderem Meetings und Projektmanagementaufgaben, die nicht spezifisch einer Iteration zugeordnet werden können.

2.4. Meilensteine

Parallel mit dem Abschluss einer Phase ist gleichzeitig ein Meilenstein erreicht. Die Bedeutung von Gitlab-Milestones und Meilensteine ist nicht deckungsgleich. Gitlab-Milestones werden von uns zum Verwalten von Issues, die in der selben Iteration geplant sind, verwendet. Ein Meilenstein kann mit dem Abschluss einer Iteration zusammenfallen, jedoch gibt es auch Iterationen, die beim Abschluss kein Erreichen eines neuen Meilenstein bedeuten.

2.4.1. Elaboration abgeschlossen

Mit Erreichen dieses Meilensteins ist der Projektplan fertig, die Definition der Anforderung, Architektur und Testmetriken fertiggestellt und die Entscheidung für ein Framework getroffen. Zieltermin für diesen Meilenstein ist das Ende der Iteration «IT3: Prototypen und Architektur».

2.4.2. Construction abgeschlossen

Mit Erreichen der End of Construction Phase ist die Finale Version fertig entwickelt und getestet. Zieltermin für diesen Meilenstein ist das Ende der Iteration «IT6: Final».

2.4.3. Transition abgeschlossen

Mit Erreichen der Transition Phase ist das Projekt beendet. Die Finale Version, eine Demo und die geforderten Dokumente wurden abgegeben. Der Zieltermin für diesen Meilenstein ist das Ende der Iteration «IT7: Transition und Abgabe».

2.5. Meetings

Jede Woche findet ein Meeting mit dem Betreuer der Studien-/Bachelorarbeit statt. Gemäss Abmachung werden die Meetings jeweils am Donnerstag um 9:00 abgehalten. Die Agenda für das Meeting wird von den Studenten mindestens 24 Stunden vor dem Meeting per E-Mail dem Betreuer mitgeteilt. Das Protokoll für die Sitzung wird ebenfalls von den Studenten erstellt und sollte ebenfalls innerhalb von 24 Stunden nach dem Meeting an den Betreuer weitergereicht werden.

Um auf die Sitzung möglichst gut vorbereitet zu sein und um sicherzustellen, dass das Entwicklerteam eine gemeinsame Meinung vertritt, trifft sich das Entwicklerteam vor den Sitzungen um sich abzusprechen.

Das Entwicklerteam trifft sich zudem an jedem Dienstag Morgen, um offene Punkte anzusprechen, Issues zu priorisieren, Arbeiten zu verteilen und die Traktanden für die Sitzung mit dem Betreuer zu erstellen.

3. Risikomanagement

Um das Risiko des Projektfehlschlags zu minimieren sind Teilrisiken im Projekt identifiziert worden. Die Risikoaufzählung umfasst die wahrscheinlichsten technischen Risiken, erhebt jedoch keinen Anspruch auf Vollständigkeit. Der abgeschätzte gewichtete Schaden der Risiken wird bei der Zeitplanung als Reserve miteinbezogen. Dieser wird aus allen berücksichtigten Risiken berechnet. Die Gesamtlaufzeit des Projekts wird um diese Reserve verlängert, ohne jedoch die Schätzung einzelner Arbeitspakete anzupassen. Die Risiken werden im Gitlab Wiki geführt und ständig aktualisiert. Der Vorteil dieser Lösung ist, dass das Wiki git-basiert ist und somit eine erleichterte Übersicht der Risikoentwicklung über die gesamte Projektlaufzeit bietet.

3.1. Umgang mit Risiken

3.1.1. Risikoprävention und Umgang

Für jedes Risiko ist eine Präventivmassnahme definiert, welche umgesetzt wird, um den Schaden des Risikos zu minimieren. Auch wurde für jedes Risiko ein Vorgehen definiert, welches im Eintrittsfall umgesetzt werden muss und helfen sollte den Schaden in Grenzen zu halten.

3.1.2. Risikoküberwachung

Um das Risikomanagement aktiv umzusetzen, werden die Risiken in dem wöchentlichem Teammeeting beobachtet und bewertet. Sollte bei dieser periodischen Analyse ein Risiko eintreffen, werden die Massnahmen zum Umgang des Risikos besprochen. Ebenfalls werden neu entstandene Risiken in die Risikoevaluation aufgenommen und bewertet.

Eine Neuevaluation des Projekts wird durchgeführt, wenn zu viele der eingeplanten oder neu entstandenen Risiken eintreten sollten.

Nr.	Title	Beschreibung
Anforderungen		
RA1	Späte Änderung der Anforderungen	Es treten Änderungen in den Anforderungen verspätet auf.
RA2	Entwicklung der falsche Funktionalität	Implementierung der falschen Funktionalität
RA3	Erstellung des falschen UI	Das UI passt nicht auf die Benutzer
RA4	Nicht erfüllen von NFR	NFR werden nicht erfüllt
Projektkomplexität		
RC1	Verwendung von dem Team unbekannter Technologie	Es wird eine Technologie verwendet die dem Team neu ist.
RC2	Verwendung von nicht ausgereifte Technologie	Die Technologie leidet noch an Kinderkrankheiten. (Bugs & Fehlende Doku)
RC3	Zu hohe Komplexität der Thematik	Die Komplexität der zu entwickelnden Software wird unterschätzt
RC4	Neue Programmiersprachen	Nicht genügend Kenntnisse über die eingesetzten Programmiersprachen (Javascript, C#, HTML, CSS)
RC5	Problem mit Frameworks	Grosse Veränderungen beim verwendeten Framework z.B. Änderung der Lizenz, neue API, usw.
Planung und Management		
RP1	Projektmanagement Tools Probleme	Software-Projektmanagement und Prozessmanagement Tools genügen nicht unseren Ansprüchen
RP2	Schlechte Projektplanung	Das Projekt ist schlecht geplant. z.B. relevante Artefakte gehen vergessen
RP3	Schlechte Teamkommunikation	Die Kommunikation im Team findet nicht im optimalen Mass statt
RP4	Projektfortschritt schlecht überwacht	Durch fehlendes Überwachen wird nicht erkannt, dass das Projekt schlecht läuft
RP5	Falsche Zeiteinteilung	Das Zeitmanagement geht nicht auf
RP6	Qualität der Software	Guidelines, Konfigurationsmanagement und Issuetracking wird nicht zuverlässig durchgeführt

Tabelle 3.1.: **Risikodefinition**

Nr.	Prävention	Verhalten bei Eintritt
RA1	Agiles Arbeitsvorgehen	Korrektur des Zeitplans
RA2	Klare Analyse und Formulierung von Anforderungen	Korrektur des Umfangs und des Zeitplans
RA3	Usability Test durchführen	Einfachere, aber an Benutzer angepasste Implementierung umsetzten
RA4	Verhinderung des Risikos durch fortlaufendes Testing und Monitoring	Anpassung damit NFR erfüllt werden wenn möglich, ansonsten Gespräch mit dem Auftraggeber
RC1	Verwendung von bekannter Technologie soweit wie möglich	Schnelle und Effiziente Einarbeitung in Technologie
RC2	Vorbeugung durch Analyse der Technologien	Wenn möglich Technologie wechseln oder externe Ressource um Hilfe bitten.
RC3	Erstellung von Prototypen, welche die wichtigsten Funktionen implementieren	Funktionalitäten streichen
RC4	Seriöses Einarbeiten in die verwendeten Sprachen mit Dokumentationen, Tutorials	Programmierkenntnisse nebenbei vertiefen
RC5	Vergleich der einzelnen Frameworks und Zukunftssicherheit als wichtiges Kriterium werten	Wenn möglich Technologie wechseln oder externe Ressource um Hilfe bitten.
RP1	Sorgfältiges Vergleichen von passenden Tools	Auf anderes Managementtools migrieren
RP2	Ausführlicher Projektplan erstellen	Nachträgliche Planung der vergessenen Komponenten
RP3	Verwendung eines Team-meetings sowie erzwungene Kommunikation durch Definition im Workflow	Einführung von zusätzlichen Kommunikationsmassnahmen
RP4	Visuallisierung und Thematisierung des Projektvorschritts	Zusätzliche Überwachungsmassnahmen einführen
RP5	Sorgfälltige Aufwandschätzung	Umfang reduzieren
RP6	Guidelines früh einsetzen und regelmässige Codereviews durchführen	Mehr Reviews durchführen, Entwickler auf die mangelnde Qualität hinweisen

Tabelle 3.2.: **Vorbeugung und Umgang mit Risiken**

Nr.	Max. Schaden	Eintrittswahrscheinlichkeit	Gewichteter Schaden
	h	%	
RA1	30	5	1.5
RA2	25	10	2.5
RA3	10	10	1
RA4	20	5	1
RC1	15	30	4.5
RC2	20	10	2
RC3	20	10	2
RC4	25	5	1.25
RC5	20	10	2
RP1	10	5	0.5
RP2	20	5	1
RP3	15	10	1.5
RP4	25	5	1.25
RP5	30	20	6
RP6	20	10	2

Tabelle 3.3.: **Risikogewichtung**

4. Infrastruktur

Jedes Mitglied des Entwicklerteams benutzt seinen persönlichen Laptop für die Entwicklung und die vorhanden und bekannten Tools um ein möglichst effizientes Vorgehen zu ermöglichen.

4.1. Gitlab

Gitlab vereint Projektmanagementtools wie Issue Tracking, Verwaltung von Milestones und Erfassung von Arbeitszeiten, ein git-basiertes Repository für den Sourcecode, Continuous Integration/-Delivery und Erstellen eines Wikis.

4.1.1. Repository

Der Quellcode des Bots, sowie die Dokumentation, welche wir in Latex machen, wird in einem gemeinsamen, privaten Gitlab Repo verwaltet.

4.1.2. Continuous Integration/Delivery

Wird von Gitlab ein neuer Push registriert, wird ein neuer Build generiert. Wenn die Tests abgeschlossen sind, wird der fertige Bot automatisch deployed. Ziel ist es, dass ständig eine lauffähige Version zur Verfügung steht.

4.1.3. Wiki

Das Wiki ist für dieses Projekt sehr hilfreich, da Issues, Milestones, Commits, Merge Requests einfach verlinkt werden können. Zudem ist das Wiki git-basiert, was den Zugriff auf ältere Versionen einzelner Wikipages ermöglicht.

Das Wiki wird für die gesamte technische Dokumentation verwendet, um Workflows (z.B. den Gitworkflow und das Vorgehen beim Time Tracking von Issues mit Gitlab und Toggl) und Fragen an den Betreuer festzuhalten und um Risiken zu verwalten.

4.2. Toggl

Leider bietet Gitlab kein detailliertes Reporting der geleisteten Arbeitszeiten auf einzelne Issues. Um trotzdem ein umfangreiches Reporting zu ermöglichen, haben wir uns für die Verwendung von Toggl entschieden. Toggl ist ein Time Tracking Tool. Mittels Browser Extension kann Toggl direkt in Gitlab eingebunden werden, so erscheint in den Issues ein Button mit dem der Toggl Timer gestartet wird.

Nach beendeter Arbeit können manuelle Änderungen in der Toggl Webapp gemacht werden.

5. Qualitätsmassnahmen

Bei der Arbeit wird auf die Hygienefaktoren (VCS/CI/CD) grossen Wert gelegt. Als Qualitätsmassnahmen werden verschiedene Tools eingesetzt. Es wird mit einem Versionskontrollsystem (VCS) gearbeitet. Ein VCS hilft, Änderungen zu protokollieren und ein gemeinsames Arbeiten am Projekt zu erleichtern.

5.1. Projektmanagement

Das Projektmanagement findet grösstenteils auf GitLab[1] statt. GitLab wird verwendet für das Issue Tracking, Verwaltung von Milestones und Erfassung von Arbeitszeiten. Um eine präzisere Arbeitszeiterfassung zu erhalten, wird zusätzlich zu GitLab, Toggl verwendet[2].

5.1.1. Workflows

Zur klaren Strukturierung von wiederkehrenden Arbeiten wurden Workflows definiert. Diese sind vom Team einzuhalten, damit die Qualität zu gewährleisten ist.

Arbeitsaufteilung Um der Zusammensetzung von Studien- und Bachelorarbeit gerecht zu werden, wird auf eine typ- und aufwandsgerechte Arbeitsverteilung geachtet. Die Verteilung ist ein laufender Prozess und findet durch Absprache im Team statt. Miteinbezogen bei der Vergabe der Arbeit werden folgende Faktoren: Persönliche Präferenzen, Wissen, und die momentane Auslastung der Teammitglieder.

Reviews Jegliche Arbeit ist stets von dem zweiten Teammitglied zu begutachten. Das hilft die Qualität, sowie den Fortschritt der Arbeit zu kontrollieren. Die Reviews werden im VCS-Workflow durchgeführt. Besteht die begutachtete Arbeit den Review nicht, wird bei einem grossen Problem der Urheber benachrichtig und über das Problem informiert. Bei einem kleinen Problem ist der Reviewer aufgefordert das Problem selbst zu beheben.

Meeting im Team Jede Woche findet ein Meeting im Team statt, um einen einheitlichen Stand zu schaffen und denn Austausch zu fördern. Es sind verschiedene Punkte zu besprechen:

- Rückblick

[1] https://gitlab.com/
[2] https://toggl.com/

- Evaluation/Neuerfassung von Risiken
- Arbeitsverteilung
- Ausblick

Meeting mit Betreuer Jede Woche findet ein Meeting mit dem Betreuer statt, welches vom Entwicklerteam geleitet wird.

- Traktanden erarbeiten und versenden
- Vorbereitung im Team auf das Meeting
- Durchführung des Meetings
- Nachbesprechung im Team im Anschluss des Meetings
- Protokoll schreiben und versenden

Versionskontrolle Für jedes Feature ist ein eigener Branch zu erstellen, der später über einen Merge-Request integriert wird. Der Workflow mit dem VCS gestaltet sich wie folgt:

1. Auschecken eines Feature Branches
2. Wechseln auf den Feature Branch
3. Auf lokalen Feature Branch commiten
4. Auf Remote Feature Branch pushen
5. Neuer Merge Request im Gitlab erstellen
6. Zweite Person begutachtet Merge-Request
7. Sofern die zweite Person einverstanden mit dem Merge-Request ist, führt sie den Merge-Request aus.

5.2. Codequalität

Zu Sicherstellung der Codequalität werden verschiedene Massnahmen getroffen. Die Formatierung und das Aussehen des Codes wird durch Code-Style-Guidlines geregelt, die Funktionsweise des Codes wird mittels Unit-Test überprüft.

5.2.1. Code-Style-Guidlines

Nach End of Elaboration werden hier Code Style Guidlines definiert, die zu den entsprechenden Programmiersprachen passen.

5.2.2. Unit-Testing

Wenn möglich wird mit Test-Driven-Development gearbeitet, um die Codequalität zu steigern. Generell ist auf eine möglichst hohe Unit-Test-Abdeckung zu achten.

5.3. Testing

Es sind verschiedene Testmetriken definiert, die die Qualität des Produkts überprüfen und sicherstellen sollen. Unter anderem wird die Usability getestet, damit das Produkt möglichst Benutzerfreundlich ist. Ebenfalls werden AI/Search Metriken wie Precision und Recall, sowie die kognitiven Fähigkeiten der erstellten Regeln getestet. Um die benötigte Zeit zum Testen der Usability zu senken, werden die Prototypen nur mittels Cognitive Walkthrough auf ihre Benutzerfreundlichkeit getestet. Ab der Alphaversion werden Usability Test durchgeführt. Diese Test sollten möglichst schlank sein und pro Test wird mit maximal fünf Personen, die sich mit Softwarearchitektur auskennen, getestet.

5.4. Versionierung

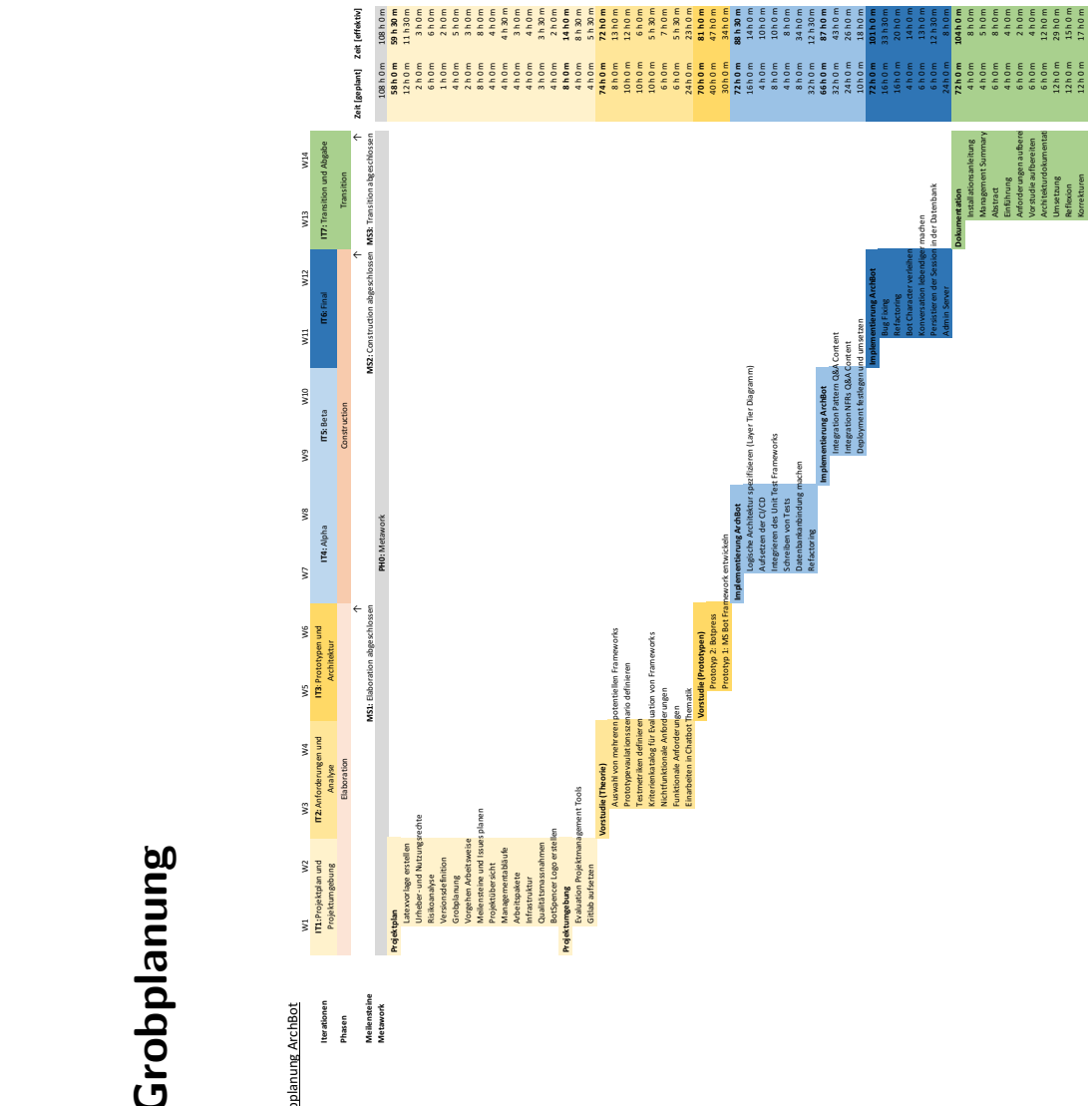
Damit die Version jeglicher Arbeit, sowie Code welcher das VCS verlässt, nachzuvollziehen ist, wird für jedes Dokument eine Versionsnummer festgelegt. Als Guideline wird «Semantic Versioning 2.0.0[3]» verwendet.

Zusammenfassung Semantic Versioning 2.0.0
Die Versionsnummer setzt sich zusammen aus: «MAJOR.MINOR.PATCH-LABEL»

- MAJOR-Version ändert sich bei inkompatiblen API Veränderung.
- MINOR-Version ändert sich bei zusätzlicher Funktionalität welche rückwärtskompatible ist.
- PATCH-Version ändert sich bei rückwärtskompatiblen Bug-Fixes
- LABEL kann verwendet werden für Pre-release oder Metadaten Bsp: «MAJOR.MINOR.PATCH-alpha»

[3] http://semver.org/

A. Grobplanung



G. Technische Risiken

Dim	Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten	Status
Anforderungen									
	RA1	Späte Änderung der Anforderungen	Es treten Änderungen in den Anforderungen verspätet auf.	30	5%	1.5	Agiles Arbeitsvorgehen	Anpassung des Zeitplans	OK
	RA2	Entwicklung der falsche Funktionalität	Implementierung der flaschen Funktionalität	25	10%	2.5	Klare Analyse und Formulierung von Anforderungen	Anpassung des Umfangs und des Zeitplans	OK
	RA3	Erstellung des falschen UI	Das UI passt nicht auf die Benutzer	10	10%	1	Usability Test durchführen	Einfachere aber an Benutzer angepasste Inplementierung umsetzten	OK
	RA4	Nicht erfüllen von NFR	NFR werden nicht erfüllt	20	5%	1	Verhinderung des Risikos durch vorlaufendes Testing und Monitoring	Anpassung damit NFR erfüllt werden wenn möglich ansonsten Gespräch mit dem Auftraggeber	Zwei konnten nicht getestet werden
Projekt Komplexität									
	RC1	Verwendung von dem Team unbekannter Technologie	Es wird eine Technologie verwendet die dem Team neu ist.	15	30%	4.5	Verwendung von Bekanntertechnologie soweit als möglich	Schnelle und Effiziente Einarbeitung in Technologie	OK
	RC2	Verwendung von nicht ausgereifte Technologie	Die Technologie leidet noch an Kinderkrankheiten. (Bugs & Fehlende Doku)	20	10%	2	Vorbeugung durch Analyse der Technologien	Wenn möglich Technologie wechseln oder externe Ressource um hilfe bitten.	Zum Teil eingetroffen (KW44), Traff für alle Frameworks zu, konnte deshalb nicht ausgewechselt werden
	RC3	Zu hohe komplexität des Thematik	Die Komplexität der zu entwickelnden Software wird unterschätzt	20	10%	2	Erstellung von Prototypen, welche die wichtigsten Funktionen implementieren	Funktionalitäten streichen	Zum Teil eingetroffen (KW46), Funktionalitäten wurden gestrichen
	RC4	Neue Programmiersprachen	Nicht genügend Kenntnisse über die eingesetzten Programmiersprachen (Javascript, C#, HTML, CSS)	25	5%	1.25	Seriöses Einarbeiten in die verwendeten Sprachen mit Dokumentationen, Tutorials	Programmierkenntnisse nebenbei vertiefen	Eingetroffen mit JavaScript, grössere Herausforderung wie angenommen (KW44)
	RC5	Problem mit Frameworks	Grosse Veränderungen beim verwendeten Framework z.B. Änderung der Lizenz, neue API, usw	20	10%	2	Vergleich der einzelnen Frameworks und Zukunftssicherheit als wichtiges Kriterium werten	Wenn möglich Technologie wechseln oder externe Ressource um hilfe bitten.	OK
Planung und Management									
	RP1	Projektmanagement Tools Probleme	Software-Projektmanagement und Prozessmanagement Tools genügen nicht unseren Ansprüchen	10	5%	0.5	Sorgfältiges Vergleichen von passenden Tools	Auf anderes Managementtools migrieren	OK
	RP2	Schlechte Projektplanung	Das Projekt ist schlecht geplant. z.b relevante Artefakte gehen vergessen	20	5%	1	Ausführlicher Projektplan erstellen	Nachträgliche Planung der vergessenen Komponenten	OK
	RP3	Schlechte Teamkommunikation	Die Kommunikation im Team findet nicht im optimalen Mass statt	15	10%	1.5	Verwendung eines Team-meetings sowie erzwunge Kommunikation durch Definition im Workflow	Einführung von zusätzlichen Kommunikationsmassnahmen	OK
	RP4	Projektfortschritt schlecht überwacht	Durch fehlendes Überwachen wird nicht erkannt wenn das Projekt schlecht läuft	25	5%	1.25	Visuallisierung und Thematisierung des Projektvorschritts	Zusätzliche Überwachungsmassnahmen einführen	OK
	RP5	Falsche Zeiteinteilung	Das Zeitmanagement geht nicht auf	30	20%	6	Sorgfälltige Aufwandschätzung	Umfang reduzieren	OK
	RP6	Qualität der Software	Guidelines, Konfigurationsmanagement und Issuetracking wird nicht zuverlässig durchgeführt	20	10%	2	Guidelines früh einsetzen und regelmässige Codereviews durchführen	Mehr Reviews durchführen, Entwickler auf die mangelnde Qualität hinweisen	OK
			Totaler Schaden			**28**			