Rene Berger, BSc

TUG Searchchatbot

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

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

Co-Advisor

Dipl.-Ing. Markus Ebner

Institute of Interactive Systems and Data Science
Head: Univ.-Prof. Dipl.-Ing. Dr. Stefanie Lindstaedt

Graz, May 2018

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Eidesstattliche Erklärung

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

Datum	Unterschrift

Abstract

Technologies changed in recent decades very often. In the past, the classic offline desktop application was the most popular one, current web and mobile applications have a large influence on the software market. For that reason, the way how to use a Graphical User Interface (GUI) changed heavily. The concepts of guiding a user through the application is widely spread, but one approach which is getting more and more popular is the chatbot. The functionality of a chatbot can be very simple by only accepting defined commands, however there are smart bots with artificial intelligence to satisfy the expectations of the user.

In this thesis a chatbot for the Graz University of Technology (TU) is designed and developed, which includes all the functionality of the Graz University of Technology search mobile application, furthermore the search result will be improved in comparison to the existing implementation. Moreover, artificial intelligence will be included to recognize certain patterns and provide better results during user interaction. The main goal of this application is to provide a text-based communication concept, that the user easily receives the desired content without using a common graphical user interface.

The procedure of the development of the chatbot application is described, starting with analyzing the state of the art technologies, which can be used on client and server side, the specific implementation, the chosen architecture and finally the evaluation of the user experience.

Kurzfassung

Technologien haben sich in den letzten Jahrzehnten immer wieder geändert. Waren es früher offline Desktop Programme, so sind es nun Web- und Mobile-Applikationen, welche große Präsenz am Softwaremarkt erlangen. Damit verbunden ergeben sich auch neue Bedien- und Interaktionskonzepte. Neben der klassischen Benutzeroberfläche, welche nur per Klick bedient werden kann, ergeben sich weitere Möglichkeiten um dem Anwender die gewünschte Information aufzubereiten. Ein solches Konzept, dass immer mehr an Bedeutung gewinnt, ist der Chatbot. Angefangen von simplen Varianten, welche definierte Befehle entgegennehmen, bis hin zu Bots mit ausgereifter künstlicher Intelligenz, sind dem Entwickler keine Grenzen gesetzt.

Im Zuge dieser Arbeit wird ein Chatbot entwickelt, welcher die Funktionalität der Technischen Universität Graz (TUG) Search Mobile-App implementiert und zudem die Suchergebnisse verfeinert um der/dem Anwender/in die bestmögliche Benutzer/innen-Erfahrung zu bieten. Des Weiteren wird der Chatbot mit künstlicher Intelligenz ausgestattet, um im Laufe der Zeit Muster von Suchanfragen besser erkennen und somit genauere Ergebnisse liefern zu können. Das Hauptziel dieser Applikation ist es der/dem Anwender/in eine textbasierte Kommunikationsform zu ermöglichen und dieser/diesem zielgerichteter und schneller gewünschte Inhalte bereit zu stellen.

In dieser Arbeit wird die Vorgehensweise für die Entwicklung der Applikation beschrieben, beginnend mit der Evaluierung der zu verwendenden client- und serverseitigen Technologien, die konkrete Implementierung, die gewählte Architektur und das Design bis hin zur Auswertung der Benutzer/innen-Erfahrung.

Contents

API	Application Programming Interface
AWS	Amazon Web Services
CLI	Command-line interface
CSS	Cascading Style Sheet
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
NLU	Natural Language Understanding
NPM	Node Package Manager
PHP	PHP: Hypertext Preprocessor
REST	Representational State Transfer
SDK	Software Development Kit
TU	Technische Universität/University of Technology
TUG	Technische Universität Graz/Graz University of Technology
XML	Extensible Markup Language

Table 1: Acronyms

List of Tables

List of Figures

Listings

1 Introduction

1.1 Motivation

The communication with a computer based system has many advantages, namely, the user is able to directly ask for the desired information. In a common graphical user interface based design concept, it is necessary to navigate through the application, going through multiple steps to achieve the same result. A communication based concept guides the user more or less until the desired content is provided. This approach is more target-oriented and natural then going through an application by using the GUI. Combined with artificial intelligence, a chatbot is a smart next generation interface alternative to present information.

1.2 Problem

The current feature set of the TU Graz search mobile app serves as a base which content should be delivered. After analyzing the mobile application, it turned out that the search results often are not satisfactory. Moreover, the developed search-bot should be easy to use and provide a concept in which the user knows immediately how to communicate with the system. Additionally, the various search topics should be presented and handled in an understandable way.

1.3 Objective

The main goal of creating the TU Graz Searchchatbot is to provide a text-based dialog system to enhance the user experience in comparison to a classic desktop or mobile application.

To reach that goal certain parts of the system were developed, taking into account that the application had to be able to communicate with the TU Graz search proxy to fetch the desired information. Furthermore there had to be an interface layer, communicating with an artificial intelligence platform, and a front-end application which interacts with the user. Finally, the user should be able to retrieve all defined information already available through the TU Graz search mobile application.

2 Chatbot

2.1 What is it?

A chatbot is an interface to a service, which interacts with the user. There are three types of bot implementations, the simple scripted have behaviours that are determined by rules; smart bots, with artificial intelligence, and finally, combined solutions. They can live in a major chat product, like Slack or Facebook Messenger, or have their own interface. In the last years bots became more popular and they are heavily used all over the internet in various industries, not only in business to consumer area, but also business to business bots are on the rise.[Shevat, 2017]

The definition chatbot often comes with misconceptions. The most typical is that the bot communicates with a user in the exact same way a human would do. In fact, this is not the purpose of a common chatbot, it may be the aim for the future, but it is simply not possible to achieve it now with the technology currently available. This would lead to unrealistic expectations and frustration. Nevertheless, it is undoubtedly some kind of revolution in the software industry, the same way mobile or web application were when they appeared. It is important to keep in mind that, as mentioned before, the bot is not the service itself, but the interface to an underlying service. Therefore, a chat bot is simply a new way to provide a service through an text-based conversational interface.

In the strict sense the first chatbot was back in the days ELIZA[Weizenbaum, 1978], built by Joseph Weizenbaum in 1966. It was the first bot which made it possible to communicate with a computer in natural language. It could run several scripts and every single script emulated a different dialog partner. The most successful one was the emulation of a physiotherapist which used a thesaurus. The key success of this bot was that it took, regarding to the most important term within a sentence of the questioner, one hypernym and built a phrase to that topic. So the bot has no understanding about the topic it was a simple lookup of topics. If the person said ”I have a problem with my father,” the bot took father as the most important term, and determined family as hypernym. So the answer was ”Tell me more about your family.” Some users did not realize that they spoke with a bot although this was not the intention of the bot as mentioned above. Joseph Weizenbaum was very surprised by that result, and also today this effect of acting like a human is called the ELIZA-effect. Of course, as already described, this only worked in this context. A real life implementation does not focus on talking like a human would do.[Luka Bradeško, 2012]

To understand why chatbots are popular it has to be noted that the software industry is continually changing, and after the initial success of the web, mobile became more important and many software providers started implementing native mobile applications. This led to the market becoming saturated and it was hard to compete, as users did not want to constantly go through the process of install and uninstall different applications. One kind of application that successfully stood out was messengers in all various types. Table 2.1. shows the usage of chat applications in 2016. Users spend most of the time, while using applications on their phone, with messengers. Therefore, the software industry saw a chance in exposing their services in this heavily used applications. In fact this was the start of the success of chatbots. So there is an evolution of interfaces from web over mobile to conversational. Some industries today even rely on the bot first approach, so every branch of business should be aware of the importance of this new way of user experience.[Shevat, 2017][Freitas and Bhintade, 2017][Klopfenstein et al., 2017]

Network	**Origin**	**Monthly active users**
WhatsApp	US	1 billion
Facebook Messenger	US	900 millions
Viber	Israel	784 million
WeChat	China	762 million
Line	Japan	560 million
Instagram	US	500 millions
Kik	Canada	275 millions
Snapchat	US	220 million (est.)
Hike	India	100 millions
Palringo	UK	40 millions

Table 2.1: Chat statistics in 2016 [Munford, 2016]

2.2 Why do people use it?

As already mentioned, messenger applications are heavily used and industries want to expose their services via chatbots. Microsoft CEO Nadella said once "Bots are the new apps" [Cava, 2016]. Furthermore, he exemplified a vision how the communication between chatbots and humans can be in the future.

> "People-to-people conversations, people-to-digital assistants, people-to-bots and even digital assistants-to-bots. That's the world you're going to get to see in the years to come" [Cava, 2016]

But why are the chatbots accepted by people and why does so many use them? In a study of [Brandtzaeg and Følstad, 2017] in 2017 this question was analysed. The goal was to find out what is the main motivation of people for using chatbots. In table 2.2 the categories extracted from the answers of the participants are listed.

The majority of the participants use as expected chatbots in terms of productivity. The reason why productivity increases when using a chatbot is mostly because of the ease of use, the speed, and of course the convenience. It saves time to ask directly what information a person wants to have, instead of browsing a webpage or use a search engine. For example waiting

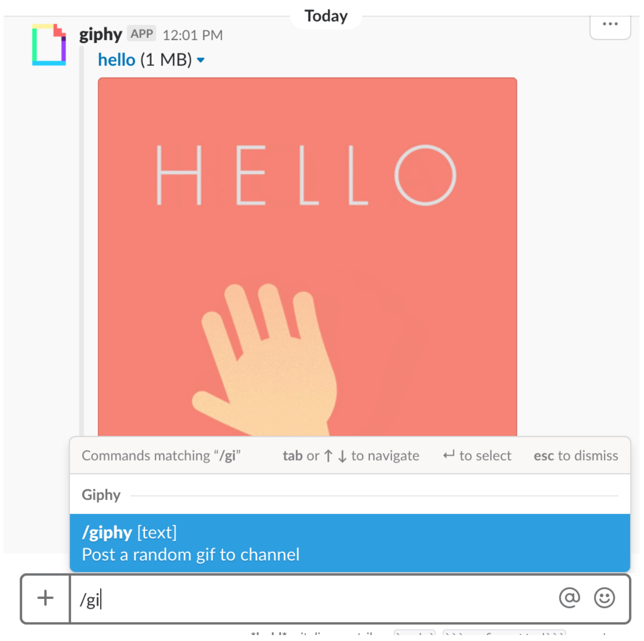

Today
giphy APP 12:01 PM
hello (1 MB)
HELLO
Commands matching "/gi" tab or ↑ ↓ to navigate ↵ to select esc to dismiss
Giphy
/giphy [text]
Post a random gif to channel
/gi

Figure 2.1: Giphy slack chatbot

on the phone to get an answer from someone is not a problem when using a chatbot. People also integrate chatbots in their daily life to execute tasks which can be automated.

Another reason is that people are entertained by chatbots. In general they like to ask questions and get entertaining answers back, get funny tips or receive rich media via chatbots. The giphy chatbot for slack is an example for that use case. Also, social and relational purposes were mentioned as a motivation. When people are bored and they have no one to talk to, a chatbot is an alternative for them, although everyone knows that the dialog partner is not real. Finally, the novelty of chatbots is also a reason for people to use them. Summarizing, it is interesting for people to try to figure out how smart they are. At the beginning, often people do not know how a certain kind of chatbot can improve their daily life, so they test them because its new, and if they get in touch with an useful bot, they integrate it in their routines, either for professional or private reasons. [Shevat, 2017]

Category	**Description**	**Frequency**
Productivity	The comment concerns the convenience of using chatbots (whether they are easy or fast to use). Participants typically report using them to obtain assistance or information.	100
Entertainment	The comment concerns the entertainment value of using chatbots (whether they are fun to use). Some report that they use chatbots when bored to kill time.	29
Social/relational	The comment concerns the use of chatbots for social or relational purposes. Typically, chatbots are seen as a personal, human means of interaction that may have social value. Some also use chatbots to strengthen social interactions with other people.	18
Novelty/ Curiosity	The comment concerns the use of chatbots out of curiosity 15 or because they are a novelty. Often, the stated aim is to investigate chatbots' capabilities.	15
Other	The comment concerns motivations that do not fit in the 12 above categories and are not sufficiently frequent to justify a separate category.	12

Table 2.2: Categories of motivation for chatbot use (N = 146). Note: 16% of the responses were coded as addressing two or more themes. [Brandtzaeg and Følstad, 2017]

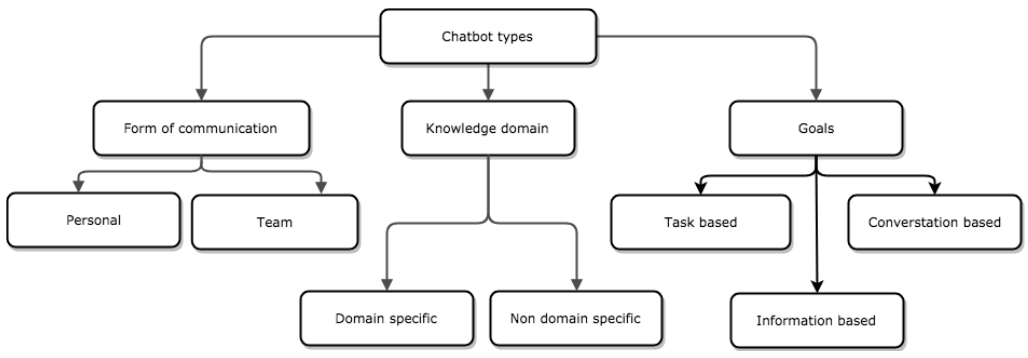


Figure 2.2: Chatbot types

2.3 Types

As shown in figure 2.2, there are several types of chatbots. Basically they are a combination of three subtypes, they can be personal or impersonal in combination with the goal and the domain knowledge. Following the characteristics of this kind of bots are described.

2.3.1 Form of communication

Personal

A personal bot has a single user approach. The main goal of this type is to satisfy the needs of an user on an one on one basis. It can be seen as some kind of a personal assistant. An example for this kind of bot would be a shopping assistant chatbot for Zalando[1] or H&M[2]. So the bot communicates with a single user over a single context.[Shevat, 2017]

[1]http://www.zalando.at, accessed 19.04.2018
[2]http://www.hm.at, accessed 19.04.2018

Team

A team bot, by contrast with the single bot, has a multiple user approach. The implementation of this type is more complex because it has to handle multiple user inputs for example in a shared channel and has to switch between conversations. This bot has to know the current context of every conversation inside of a channel. Set up meetings is a typical application area for this kind of bot.[Shevat, 2017]

	User basis	**Characteristics**	**Example**
Personal	one on one	single context, personal assistant	Zalando
Team	one to many	multiple context, used in shared channels	Slack

Table 2.3: Comparison personal versus team bot

2.3.2 Knowledge Domain

Domain specific

A domain specific bot is typically implemented for a single service or product and the user associates this bot with it. As mentioned above, the H&M personal shopping bot fits in this category. It handles only shopping related actions and it represents the brand H&M. An advantage of domain specific bots is that it can specialize for one topic. It can also be designed with high customization, so the development does not need to take into account standardising the bot.[Nimavat and Champaneria, 2017]

Non domain specific

Non domain specific bots are sometimes referred to as super bots. Unlike the domain specific ones, they expose multiple services. A very well known super bot is Amazon's Alexa[3], which handles a huge amount of services. The advantage of these type of bots is that the user only needs to interact to a single interface instead of having to find one for every topic. There is no need to learn how a bot for a specific service works, everything is standardised. The disadvantage for the industry is that there is less control about how your service is exposed. The super bot is responsible for handling the user experience.[Nimavat and Champaneria, 2017]

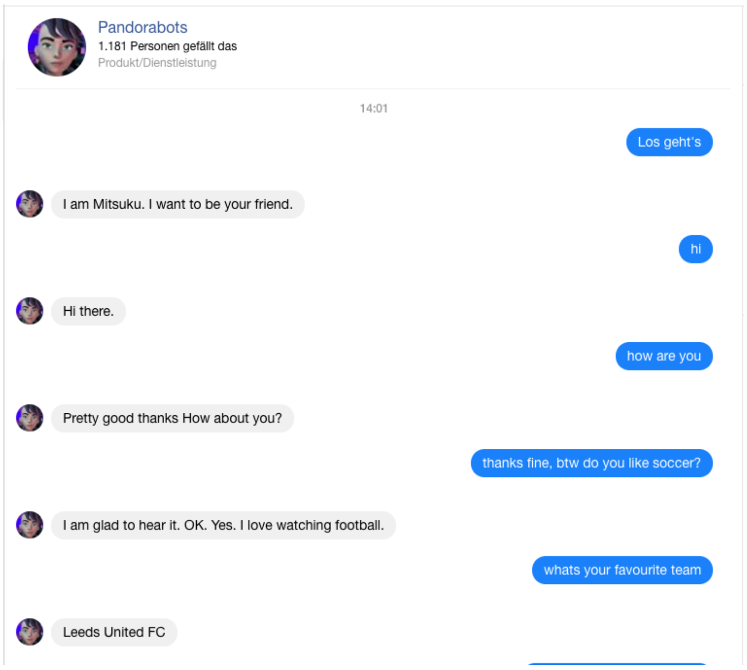
	Services	**Characteristics**	**Example**
Specific	single service	specialised, represents a brand or product	H&M
Non specfic	multiple services	standardised, less control of services handled by the bot	Alexa

Table 2.5: Comparison domain specific versus non domain specific bot

2.4 Goals

2.4.1 Task based

This kind of bot is implemented to execute a certain task. The flow of this conversation is predefined in most cases with the goal to finish the job. An example for that is a shopping bot. It is intelligent in a way to ask and understand the purpose of the user, for example, the shopping bot would ask for the type of clothes the user wants to buy and refines the questions until it is clear what the user wants to buy.[Nimavat and Champaneria, 2017]


Pandorabots
1.181 Personen gefällt das
Produkt/Dienstleistung

14:01

Los geht's

I am Mitsuku. I want to be your friend.

hi

Hi there.

how are you

Pretty good thanks How about you?

thanks fine, btw do you like soccer?

I am glad to hear it. OK. Yes. I love watching football.

whats your favourite team

Leeds United FC

Figure 2.3: Mitsuku chatbot

2.4.2 Conversation based

These bots are trying to communicate with the user in the more natural way. There is no certain task or action that they want to execute, the conversation itself is the goal. To achieve that, it is necessary to understand the users input very well and, because of that its heavily based on artificial intelligence. Mitsuku and Siri are examples of this kind of approach.[Nimavat and Champaneria, 2017]

2.4.3 Information based

Information based bots provide content for specific topics. The conversation flow should be short and should deliver the content immediately. An example for this kind is a Frequently Asked Questions (FAQ) bot. It could, for

[3]https://developer.amazon.com/de/alexa, accessed 24.04.2018

instance, parse a frequently asked questions page of a website and provide the content as a message.[Nimavat and Champaneria, 2017]

2.5 Use cases

This section presents several common use cases of chatbots. The most important one in terms of monetization is the conversational commerce. It has a big potential and industries make use of it to offer their products through chatbots. Every major e-commerce platform already launched a chatbot.[Shevat, 2017]

2.5.1 Entertainment

The purpose of this bot is to entertain the user in some way. There are different strategies to reach that, but the goal for the bot is to try to turn the conversation into entertainment. A very popular fun and entertainment bot is Swelly[4]. It provides instant community feedback about two possible options. It is available as native Android and iOS application. Celebrities use bots to stay in contact with their fans and entertain them for community engagement and raise their popularity, which could also be seen as a brand bot. Katy perry, for example, launched a messenger chatbot which won awards on some bot platforms like chatbottle[5].

2.5.2 Customer Service and FAQ

This is the most common use case. The intention of the companies is to use the bots for the first level customer support. The TU Graz Searchchatbot, covered in this thesis, fits in this category. Common tasks or questions should be handled via the bot. The TU Graz Searchchatbot provides many information about the university life. Big companies, with a lot of costumer

[4]https://www.swell.wtf/, accessed 19.04.2018
[5]https://chatbottle.co/, accessed 19.04.2018

queries, are presented with the financial decision of comparing the costs of developing a first level support bot with the monthly salary of a humans.

2.5.3 Conversational commerce

Conversational commerce is an exciting topic in general. It focuses on simplifying the traditional commerce. Ordering a product via a simple conversation is much more comfortable then searching in a web platform. Conversational commerce is growing quickly, specially the voice bots. Every mobile operating system has its own voice assistant: Apple provides Siri, Google developed Google Assistant and Microsoft has Cortana. Amazon launched Alexa which is also a personal assistant voice bot. It creates new opportunities for business, as the customer is able to use the conversational interface while preforming other activities. The user is not interrupted and has only to invest a minimum of time and attention during communication with a commerce service. A bot outside of this category is the chatbot of LEGO.

2.5.4 Health

Chatbots also arrived in the health-care sector. The influence is not as relevant as in other sectors. The intention of this kind of bot is to support its users with health related information. There is one well known bot called Super Izzy, which is about reproductive health. In this topic where there are often misconceptions and different non-scientifically based opinions, people are often also not comfortable with asking questions. Therefore, the bot aims to help its users by clarify information and by, for example, providing information about contraception, such as the birth control pill and what to do if the person forgot to take it.

2.5.5 Artificial intelligence

Artificial intelligence is an interesting topic in combination with chatbots. Many bots are using it to improve the conversation and through that to enhance the user experience. There are some artificial intelligence tools, such as dialogflow[6] or wit[7], which will be evaluated in chapter 3.3. Those solutions aim to provide an easier way to integrate this feature into a chatbot. The most promising bot in this category is Mitsuku[8] developed by Steve Worswick. This bot won three times the Loebner-prize[9], in which it has to face a Turing test. A competition specially for artificial intelligence. This test was introduced by Alan Turing in 1950 and it was designed to determine whether or not machines are able to think in way equivalent or indistinguishable from humans.[Luka Bradeško, 2012]

This test works as shown in figure 2.4. One person engages conversation with two different partner, one is a bot the other one is a human. If the questioner result is not able to distinguish which of the dialog partners is a bot and who is a human, the touring test is considered successful. So far, no bot did pass the touring test but Mitsuku reached a score of 90%. In the beginning of the nineties the dialog was about five minutes for a chosen topic. Currently, the dialog lasts twenty-five minutes and every topic is allowed. This shows the incredible progression of artificial intelligence. There are also other tests to measure the capability of the artificial intelligence. Two very well known tests are the Lovelace and the Metzinger tests. The first proofs not only the dialog flow, but it also measures the creativity of a bot. The latter expects that the bot takes part in a discussion using his own arguments, as well as cause them in the follow up conversation.

There has been criticism about bots only trying to pass the touring test because it has no importance for the industry.

[6]https://dialogflow.com/, accessed 19.04.2018
[7]https://wit.ai/, accessed 19.04.2018
[8]https://www.pandorabots.com/mitsuku/, accessed 19.04.2018
[9]http://www.aisb.org.uk/events/loebner-prize, accessed 24.04.2018

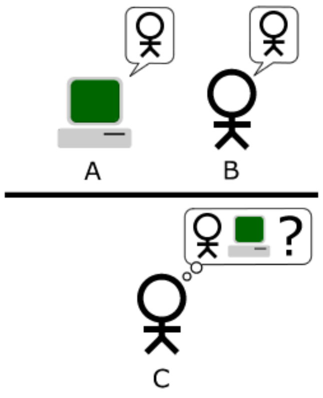


Figure 2.4: Turing test [Bilby, 2008]

2.6 Business versus consumer bot

The described types of chatbots in the section 2.3 can be applied for different realities, for both business to business and business to consumer bots. In general, those bots have different objectives. A business bot focuses on getting things done, so the conversation should be short and tasks should be dispatched very easily. An example for that is a slack bot for the project management software JIRA[10]. This bot informs you about your JIRA[11] tickets by just mentioning an issue in a channel where the bot is invited.

A consumer bot has a different conversation style. It can be more personal, off topic or just entertaining. This does not mean that there is not a goal that should still be achieved be, but the objective of this kind of bot is often assuring that the user stay in touch with the brand, or gets the most up-to-date information about a certain topic, so it can be chatty as well. In other words, they are not as task orientated as business bots.

There are two major platforms which emerged as a perfect world for chatbots. The best and widely used messenger platform which is perfect for the business to customer approach is the Facebook Messenger[12]. It is available

[10]https://slack.atlassian.io/, accessed 19.04.2018
[11]https://de.atlassian.com/software/jira, accessed 19.04.2018
[12]https://de-de.messenger.com/, accessed 19.04.2018

for mobile and for desktop devices and it provides an application programming interface for bot interaction. It is a perfect way to get in touch with possible consumers. On the other side, there is Slack[13]. It is the most known business messenger for business use cases, and it is also available for mobile and desktop. It is very common to integrate business bots for tasks like time tracking or project management support.

There are also other messengers which are worth mentioning, like Kik[14], or legacy systems like e-mail or SMS. Specially e-mail is a highly used communication option, and it is also possible to integrate the usage of chatbots. There are several provider who are specialised on supporting email chatbots. The disadvantage of a legacy platforms like e-mail is, that the set of functionalities is limited in comparison to a modern messenger, which is providing an application programming interface. IMAP and POP3 have only a very basic feature set like sending and receiving e-mails, so it is harder to build a bot with a satisfactory conversational interface.[Shevat, 2017]

2.7 Conversational Interface

The conversational interface is the core part of a chatbot. In the past, a user had to adapt to a graphical interface when using an application. User experience designers thought about how to design a program so that it can be used intuitively. However, the user had to learn which button has to be clicked for a certain action. With the conversational interface it is the opposite approach. The user knows how to use messaging applications, it is a very natural way to communicate. The interface should be designed in a way that it is as natural as possible, so that the user does not need to change his behaviour by using a messaging application. To ensure this, it is important to analyse the onboarding stage, as well as the possible strategies that can be use for dialog flow and how to handle errors.[Shevat, 2017][Crangle, 1997]

[13]https://slack.com/intl/de, accessed 19.04.2018
[14]https://www.kik.com/, accessed 19.04.2018

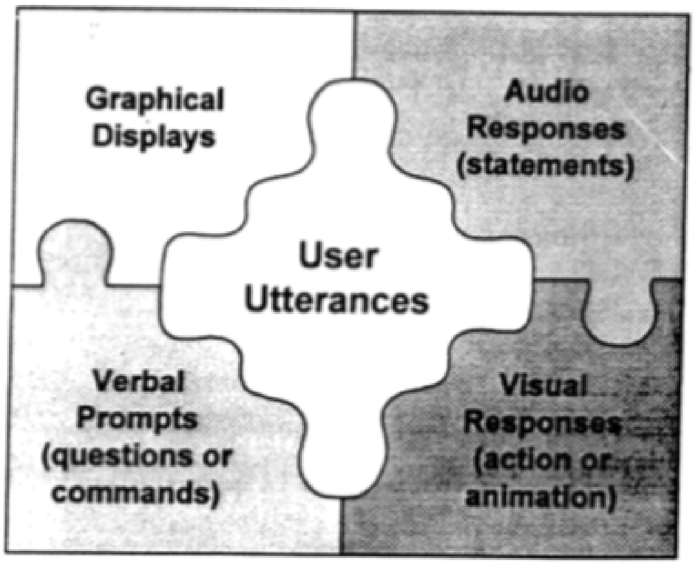

Graphical Displays
Audio Responses (statements)
User Utterances
Verbal Prompts (questions or commands)
Visual Responses (action or animation)

Figure 2.5: Types of bot interactions [Crangle, 1997]

In general it can be said that a bot, in terms of a conversation interface, is able to contribute to a conversation with four types of interactions. Namely, they are the graphical display, which will be discussed in this theses, audio responses, verbal prompts, and visual responses such as actions or animations as figure 2.5.[Crangle, 1997]

2.7.1 Onboarding

A central part of creating a conversational interface is the onboarding. In this period, the most important goal is to assure that the user knows, without hesitation or effort, what the bot is able to do or what the user can ask. The user should be engaged and informed after reading the onboarding message so that there is no confusion what the purpose of the bot is. In this phase, the user should be informed on how to interact with the bot. This is fundamental for a successful conversation, because at this time, specially if the user talks for the first time to the bot, the full attention of the user is

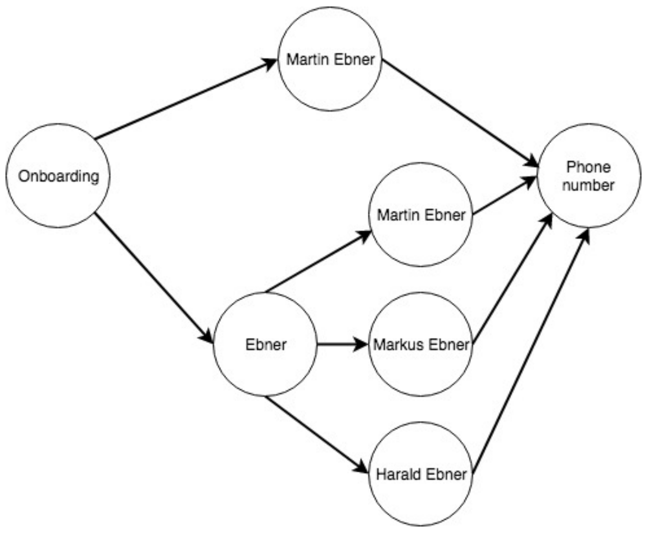


Figure 2.6: Task based conversational interface

available. If the bot has the possibility to to be configured, it should also be done in a very early stage, so the user has a clear idea of its benefit. After a successful onboarding, the user should be incited to interact, that the user gathers experience and gets an idea of the advantages of interacting with the bot. It seems evidently that the user has an idea of what can be said and how to interact but in fact it is very hard to achieve that especially in the prototype stage. [Shevat, 2017][Crangle, 1997]

2.7.2 Dialog strategies

The way the dialog flows has to be defined in the process of building a bot. This can be done with two different approaches. The first one is the task based strategy. It is very directed to execute a single task. This kind of conversation design is very common for business bots. So the intention of this approach is that the conversation is very short and goal driven. Figure 2.6 shows an example of a task based dialog. The intention is to get a phone number. The chat starts with the onboarding, the goal is the phone number of a person. This can be reached directly if the name of the person is unique or with another intermediate stage if the input has to be concretised.

The second dialog strategy is the topic based. This conversation design is not as directed and goal driven as the task based approach. It is roughly circular, and not focusing on a task that has been executed. Bots with this strategy are often used in terms of entertainment. An example would be the game of thrones bot[15], which knows everything about the series. It should keep the user informed with the topic and related news.

2.7.3 Error handling

The error handling is very important for a conversation with a chatbot. At any time during a dialog, there can be some conversational divergence. It can happen for various reasons, from a simple mishappening to the user testing how the chatbot reacts. It is the responsibility of the bot to to keep the chat ongoing, and the user engaged. There should be always a fallback for misleading questions. For example:

> User: ”asdf asdf”
> Bot: ”Sorry I do not know what you mean. Can I help you with something else?”

There is a clear statement of the bot that it can not answer the current question. There can be, certainly, other errors during a dialog flow for example off topic questions, but still there has to be always a way to keep the dialog ongoing. Restarting the conversation should be the last option.

[15]https://www.facebook.com/GoTBoT/, accessed 19.04.2018

3 TU Graz Searchchatbot

3.1 Definition of the task

The work preformed during this thesis had the aim to develop a search chatbot for the TU Graz website. Basically, it should cover the features of the already existing TU Graz search mobile application. Furthermore, the chatbot has to be integrated with the current website and should not only exists within a messenger application, like Facebook messenger, Slack, or others. For these reasons, a standalone client messenger had to be developed. It had to be taken into account that the design of this client should be similar to current messaging applications, so that there is no confusion how to use it. The client had to be responsive and easy to use also on a mobile device. For evaluation purposes, the chatbot had to be published as a part of digitallabs[1].

For prioritising the requirements, it was used the MoSCoW method [Stapleton, 2002], which is popular for using meaningful words as labels. They allow to have a clear picture of the objectives of a given project, as presented below:

- M - *MUST have this* - Represents a requirement that is essential for the project to be considered successful.
- S - *SHOULD have this if at all possible* - An item of high priority and that if at all possible should be in the included in the solution.
- C- *COULD have this if it does not affect anything else* - A requirement that is desireable to get, but that is not fundamental and must be included only if the time allows it.

[1]https://digitallabs.tugraz.at/landingpage/index.html, accessed 19.04.2018

- W - *WON'T have this but would like in the future* - Something that is out of the current scope, but should be considered for a future iteration or release.

Following the range of functions are prioritised with the MoSCoW method:

Must have

- A full responsive standalone web application chat interface
- Receive the following relevant contact information about a person which is in some working relation to TU Graz:
  - Phone number
  - E-mail address
  - Website
  - Address
- Receive information about a course at TU Graz
  - Examination date
  - General course details
- Receive information about a room at TU Graz
  - Address
- Receive information about a book at TU Graz library
  - The availability of the book
- Receive information about an organization at TU Graz
  - Phone number
  - E-mail address
  - Webite
  - Address
- Text based chatbot experience
- Meet the TU Graz design and corporate design guidelines

Should have

- Improve search results in comparison to the TU Graz search mobile app
- Provide rich media content like Google Maps for an address response

Could have

- An external artificial intelligence platform to support intent recognition with high precision.
- Support machine learning
- A general site search of the TU Graz content pages

Won't have

- Because of the need of a standalone version there is not integration in a standard messenger as already mentioned.
- Voice recognition

3.2 Architecture

The chatbot application is a full stack standalone web solution. Due to the requirements gathered, it was not possible to use an existing messenger platform. Therefore, a separate client had to be developed. In that case it had to be a single page application to prevent page loads while using the chat interface. This client communicates with a back-end, which talks to the TU Graz search proxy for receiving TU Graz related information. The bot will integrate a third party Natural Language Understanding (NLU) platform to support artificial intelligence, following NLU, and machine learning. To support that the back-end is some kind of a middleware, which interacts not only with the TU Graz search proxy, it has also to communicate with a NLU platform.

As a result of this the application consists of four main parts which are:

- Single Page Application Client
- Back-end/Middleware
- TU Graz search proxy
- Third party NLU platform for artificial intelligence support

The following example should help clarifying the outline how the TU Graz Searchchatbot was implemented. Assuming that a user wants to know the phone number of a person e.g. Martin Ebner, working at the TU Graz, the flow would be as follows:

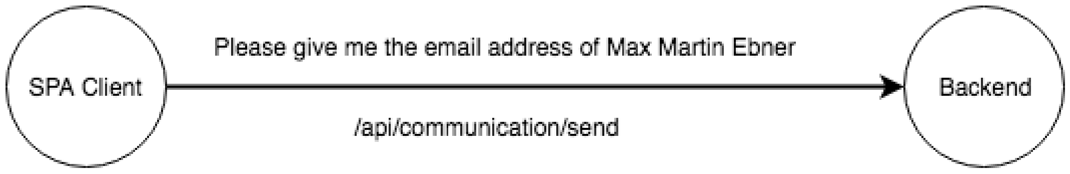
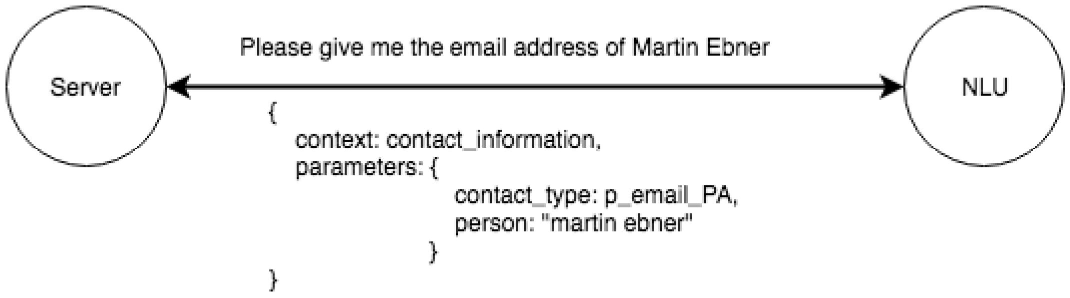


Figure 3.1: Communication between Client and Server



```
{
    context: contact_information,
    parameters: {
                    contact_type: p_email_PA,
                    person: "martin ebner"
                }
}
```

Figure 3.2: Communication between Server and NLU platform

The user starts the chatbot client by visiting the webpage and receives a session id. After that, the user sends a message *"Please give me the email address of Martin Ebner"*. The single page application client passes the session id and the message to the back-end as shown in figure 3.1.

For evaluating the intent of the user, the message has to be passed to the third party NLU tool. This platform parses the message based on the intent and extracts data, such as the context and some important variables for the given search. In that case, the context is to get contact information and the necessary parameters for the TU Graz search proxy are Martin Ebner and the contact type, which is phone number in that case as shown in figure 3.2. This information is returned to the back-end in an easy to use JSON-Object.

After receiving this data the search with the TU Graz search proxy is performed. For this example the query is:

http://search-proxy.tugraz.at/search/onebox/Search.php?query=martin%20ebner&ws=prs

The search proxy responds with a list of the found persons in a XML data format as shown in listing 3.1. The next step is to parse the XML response

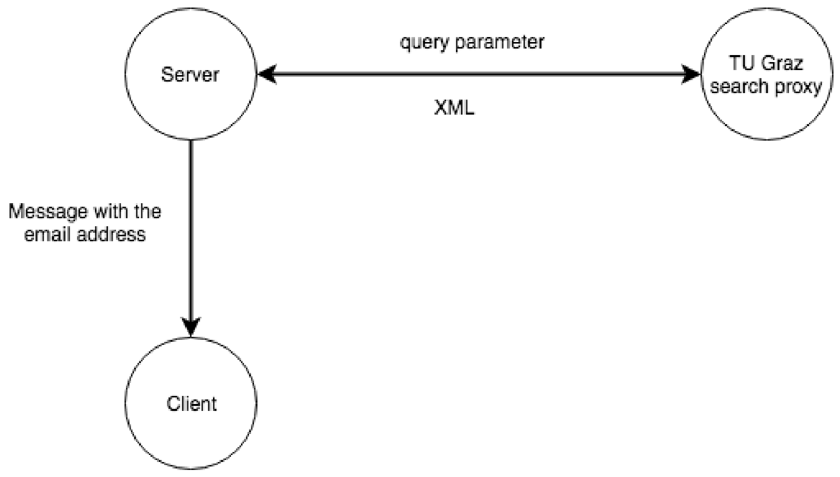


Figure 3.3: Server response to the client

to extract the desired information, which is the e-mail of Martin Ebner. The field name with the requested data is **p_email_PA**, so there should also be a mapping between the context and the field names of the XML result. After parsing the extracted information gets back to the client and the bot responds with a message as shown in figure 3.3.

```
<MODULE_RESULT>
<Title>17</Title>
<Field name="personID">99E141532528D1D7</Field>
<Field name="orgUnitID">24849</Field>
<Field name="name">Doctoral School Informatik</Field>
<Field name="userDefname">Doctoral School</Field>
<Field name="contactName">Doctoral School Informatik</Field>
<Field name="p_contact_name_PA">Lehr- und Lerntechnologien</
    Field>
<Field name="p_extadr_PA">1.Obergeschoss</Field>
<Field name="p_street_PA">Münzgrabenstrasse 35A, 1.Obergeschoss
    </Field>
<Field name="p_locality_PA">Graz</Field>
<Field name="p_pcode_PA">8010</Field>
<Field name="p_tel_office_PA">+43 (316) 873 - 8540</Field>
<Field name="p_email_PA">martin.ebner@tugraz.at</Field>
<Field name="p_tel_mobile_PA">+43 (0) 664 / 60 873 8540</Field>
<Field name="p_webLink_PA">www.martinebner.at</Field>
<Field name="p_detail_infoBlock_webLink">
```

```
https://online.tugraz.at/tug_online/visitenkarte.show_vcard?
    pPersonenGruppe=3&pPersonenId=99E141532528D1D7
</Field>
<Field name="street">Rechbauerstraße 12</Field>
<Field name="locality">Graz</Field>
<Field name="href">
http://portal.tugraz.at/portal/page/portal/TU_Graz/
    Studium_Lehre/Studien/Doktoratsstudien/Informatik_DS
</Field>
<Field name="CAMPUSonlineURL">
https://online.tugraz.at/tug_online/wborg.display?pOrgNr=24849
</Field>
<Field name="personName">Martin Ebner</Field>
<Field name="title">Priv.-Doz. Dipl.-Ing. Dr.techn.</Field>
<Field name="role">Mitglied mit Lehrbefugnis</Field>
<Field name="visit_hour">nach Vereinbarung</Field>
<Field name="tel_mobile">+43 (0) 664 / 60 873 8540</Field>
<Field name="tel_office">+43 (316) 873 - 8540</Field>
<Field name="fax_person">+43 (0) 316 / 873 - 7699</Field>
<Field name="email_person">martin.ebner@tugraz.at</Field>
<Field name="webLink_person">www.martinebner.at</Field>
<Field name="infoBlock_webLink">
https://online.tugraz.at/tug_online/visitenkarte.show_vcard?
    pPersonenGruppe=3&pPersonenId=99E141532528D1D7
</Field>
<Field name="infoBlock_pic">
https://online.tugraz.at/tug_online/visitenkarte.showImage?
    pPersonenGruppe=3&pPersonenId=99E141532528D1D7
</Field>
<Field name="personBibtexCount">681</Field>
<Field name="WEB SERVICE">PRS</Field>
<Field name="wb_result_cnt">17/141</Field>
<Field name="count_all_results">141</Field>
</MODULE_RESULT>
```

Listing 3.1: TU Graz search proxy XML result

After the basic conceptional phase the technology has to be chosen. Several NLU tools do not support a PHP SDK, so the client has to be a single page Javascript application. Due to that and the relevant support of Node.js[2] SDKś in combination with NLU Tools, this is a perfect match. The front-end

[2]https://nodejs.org/en/, accessed 24.04.2018

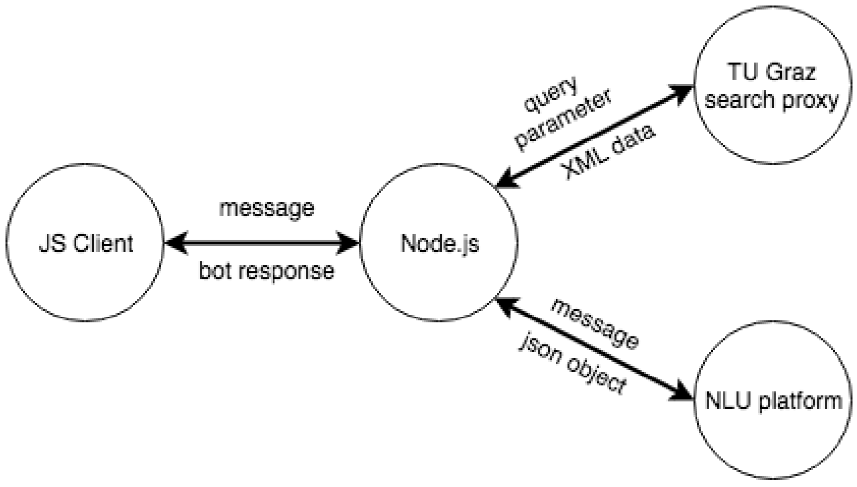


Figure 3.4: Final architecture

and back-end share the same programming language, which is a benefit for maintainability because there is no need to learn another language. Based on that decision figure 3.4 shows the final architecture.

3.3 Evaluation of NLU Tools

To support a flawless dialog flow, the intent of a message should be detected with a high precision. Therefore some external tools and platforms exists which will be described in this section. Following the best-known natural language understanding tools are listed.

dialogfow.ai

Url:	http://www.dialogflow.ai
f **Price:**	Free standard edition and a premium edition as pay as you go service
Interface:	HTTP REST API

Dialogflow[3], former api.ai, was launched in 2010 and acquired by Google in

[3]https://dialogflow.com/

2016. It parses the query and determines the most suitable intent based on the information stored in the intent. The API returns a JSON object with the calculated data. Dialogflow supports a software development kit for nearly every programming language. The intent determination works with high precision. After the fusion with Google it became possible to integrate the bot agent with actions on Google, so that applications for Google Assistant can be developed. There are predefined intents like small talk, weather, and other common dialog topics. Another available feature is the possibility of using machine learning.

wit.ai

Url:	https://wit.ai
Price:	Free including for commercial use
Interface:	HTTP REST API

Wit.ai was launched in 2013 and acquired by Facebook in 2015. It supports entities, intents, actions, and context and uses natural language processing. A key feature of wit is the support of fifty languages. For that reason, if a bot is to developed with the requirement of supporting several different languages, this platform may be the best option. Additionally, it supports software development kits for several programming languages.

recast.ai

Url	https://recast.ai
Price	Free for personal use. Individual pricing for business solutions
Interface	HTTP REST API

Recast.ai was launched in 2016. It supports software development kits for several programming languages. It uses conversational natural language processing. With an easy to use bot builder, it is also possible for people with no programming background to create and set up a bot. In comparison to wit and dialogflow, there is a smaller supporter community, although there is the possibility to share bot templates. It also supports a wide variety amount of languages, but the feature set available to them is not the same for all of them. The intent recognition is not as developed when compared to other NLU platforms.

IBM Watson conversational service

Url	https://www.ibm.com/watson/services/conversation/
Price	Pay as you go
Interface	HTTP REST API

IBM Watson conversation service was launched 2016 and it is the most promising NLU tool so far. It is built on a neural network and consists of three components. Those components are entities, intents, and dialogs. In comparison to other tools, watson supports a smaller set of SDK's. Furthermore, the language support is only for English and Japanese. Another downside is that there is no possibility to use it for free.

Microsoft LUIS

Url	http://www.luis.ai
Price	Pay as you go
Interface	HTTP REST API

Microsoft's language understanding intelligent service was launched in 2015. It uses entities and intent and returns an easy to use JSON with all necessary data. It is a tool designed for business solutions and it follows a domain specific approach, other than wit or dialogflow. A benefit of LUIS is the possible combination with Cortana. Similarly to IBM's Watson there is no possibility to use it for free.

Amazon Lex

Url	https://aws.amazon.com/lex/
Price	Pay as you go
Interface	HTTP REST API

Amazon Lex was launched in 2016 and it is also a very promising platform. Currently, Lex is limited to English, which is a disadvantage to the other platforms. It supports advanced deep learning functionalities, natural language processing and automatic speech recognition. Another feature is the easy integration to other services on the AWS platform. Also with amazon lex there is no possibility to use it for free.

In general, those platforms can be divided into business and consumer tools. Dialogflow, wit, and recast are free to use and meet basically all requirements for this thesis, although Lex, Luis, and Watson conversational service are also very interesting and promising but they have the disadvantage of their pricing model and have a business solutions background. Dialogflow has a rich feature set and it is the platform which is tested over eight years now and delivers very good results. With Google there is a big player on board and an ongoing development process is ensured. Also the integration into the google ecosystem is very interesting. Based on that dialogflow is the NLU tool used for the TU Graz Searchchatbot.

3.4 Client

In this section the evaluation of the currently most commonly used front-end frameworks is discussed. After the evaluation and the decision a proper design for the TU Graz Searchchatbot according to the requirements is created. Finally, there is a walk-through of the implementation of the application.

3.4.1 Evaluation of Front-end Frameworks

General comparison

Javascript frameworks are very popular. In the past, server side rendered applications were best practice and Javascript was only used for animations or trivial logic parts of a website. Although, there were frameworks generally used like Ember, or Backbone, it was with the appearance of Angular and React that the community started paying more attention to single page applications. With the Javacscript Frameworks, business logic from the server

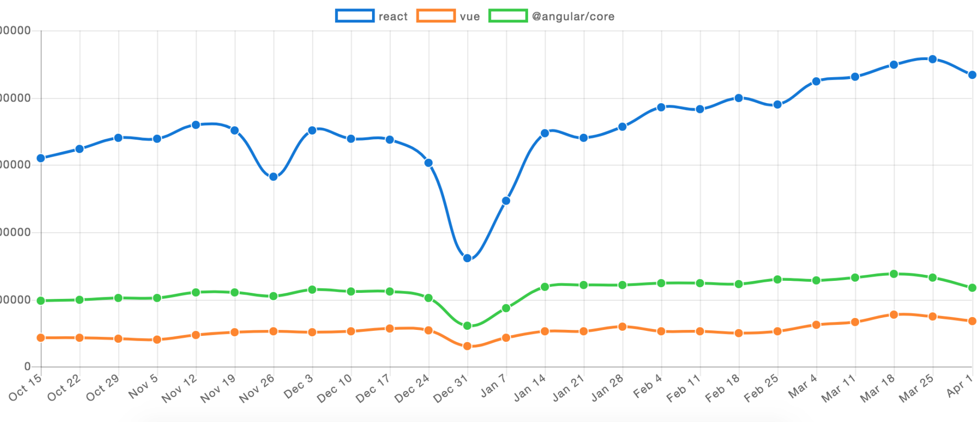


Figure 3.5: Downloads of front-end frameworks in the last six months. [Source: npmtrends[4]]

moved to the browser and Javascript became more and more important. At the moment there are three frameworks which have a communities that stand out for their size, with two of them being actively supported by two of the major players of the industry: Angular which is led by Google, React led by Facebook and Vue.js. Figure 3.5 shows the downloads via the Javascript package manager npm, that were registered in the last six months. Those three frameworks are candidates for the TU Graz Searchchatbot, although referring to the download statistics React is the clear winner. Every framework has its benefit, and they have different basic approaches.

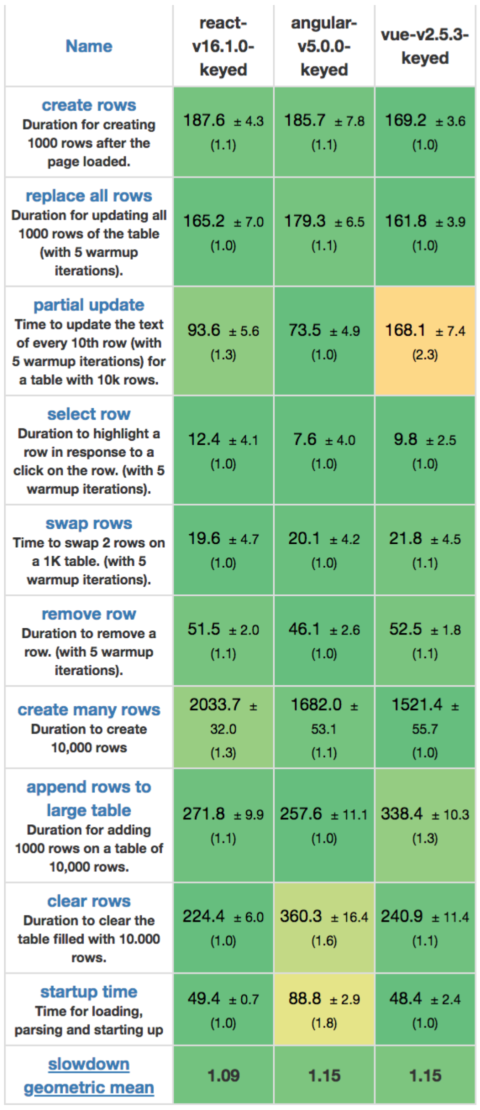
Even though they use it in slightly differently, all the three frameworks, React, Angular, and Vue share the component based approach. Apart from those, Angular also provides modules, which are wrappers for components. The same could be achieved with container components in React, but to be precise they are a in a certain extent different in their behaviour. Container components wraps functionality of a specific feature and includes other components corresponding to it. Additionally, Angular separates the logic more than the other two frameworks. There are also services, which handle all the business logic, meaning only the representational logic should be in the components. There is a similar pattern to that in React and Vue, but there still are no different classes for that.[Kunz, 2016]

[4]http://www.npmtrends.com/react-vs-vue-vs-@angular/core, accessed 19.04.2018

It is important to mention that there is also a different usage of Javascript for Angular and the other two frameworks. Angular is based on Typescript which brings type safety to Javascript, and it enjoys great popularity. Vue and React are based on plain Javascript, but there are also side project which supports a Typescript version of them. React is in comparison to Vue and Angular not really a framework, it is more a library which can be extended to a framework with third party libraries. This is also the biggest downside of React, there is no self-contained tool-set to provide full framework features. So there is always a dependency to other libraries.[Banks and Porcello, 2016]

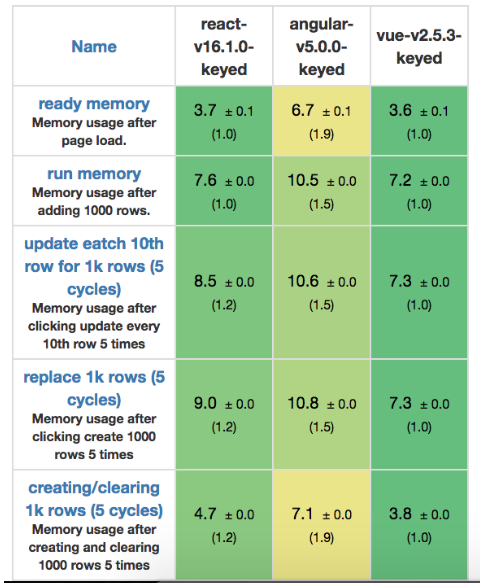
The biggest advantage of React is its rendering performance as figure 3.6 shows. It has without any doubt the fastest rendering technique. On the other hand, Angular convinces with a wide variety of tools, beginning with its own command line interface to nearly everything a developer may need when developing an application. Hence, it is marginally slower in terms of rendering, but both easy to use and stable. In comparison to AngularJS, the former version of Angular, the framework already made a step forward concerning the rendering speed. Even if there is a very complex page to render, there will not be a problem for the framework to deal with that. Vue is somewhere in between React and Angular. On one hand, it is super lightweight, but it is not a library like React, as it has all basic framework tools on board like a router, state management, and even a command line interface. Angular and Vue also support CSS modules out of the box, which is very convenient for development. This means that the styling of a component can be scoped, and will not effect other components. The natural consequence of this, is that during development there is no necessity to to take into account the naming of CSS classes and how it will affect other parts of the application. This not only increases the speed of development, but also improves maintainability and decreases the complexity for someone new needing to use the code base. React does not support that out of the box, but it is possible to configure it through the use of Webpack[5]. Webpack is an open-source module bundler, a powerful tool when working with Javascript frameworks. It loads modules bundles it and handles everything in between. All three frameworks use are using Webpack so there is no difference on that.

[5]https://webpack.js.org/, accessed 19.04.2018



Name	react-v16.1.0-keyed	angular-v5.0.0-keyed	vue-v2.5.3-keyed
create rows Duration for creating 1000 rows after the page loaded.	187.6 ± 4.3 (1.1)	185.7 ± 7.8 (1.1)	169.2 ± 3.6 (1.0)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	165.2 ± 7.0 (1.0)	179.3 ± 6.5 (1.1)	161.8 ± 3.9 (1.0)
partial update Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows.	93.6 ± 5.6 (1.3)	73.5 ± 4.9 (1.0)	168.1 ± 7.4 (2.3)
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	12.4 ± 4.1 (1.0)	7.6 ± 4.0 (1.0)	9.8 ± 2.5 (1.0)
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	19.6 ± 4.7 (1.0)	20.1 ± 4.2 (1.0)	21.8 ± 4.5 (1.1)
remove row Duration to remove a row. (with 5 warmup iterations).	51.5 ± 2.0 (1.1)	46.1 ± 2.6 (1.0)	52.5 ± 1.8 (1.1)
create many rows Duration to create 10,000 rows	2033.7 ± 32.0 (1.3)	1682.0 ± 53.1 (1.1)	1521.4 ± 55.7 (1.0)
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	271.8 ± 9.9 (1.1)	257.6 ± 11.1 (1.0)	338.4 ± 10.3 (1.3)
clear rows Duration to clear the table filled with 10.000 rows.	224.4 ± 6.0 (1.0)	360.3 ± 16.4 (1.6)	240.9 ± 11.4 (1.1)
startup time Time for loading, parsing and starting up	49.4 ± 0.7 (1.0)	88.8 ± 2.9 (1.8)	48.4 ± 2.4 (1.0)
slowdown geometric mean	1.09	1.15	1.15

(a) Duration in milliseconds +/- standard deviation (Slowdown = Duration / Fastest)



Name	react-v16.1.0-keyed	angular-v5.0.0-keyed	vue-v2.5.3-keyed
ready memory Memory usage after page load.	3.7 ± 0.1 (1.0)	6.7 ± 0.1 (1.9)	3.6 ± 0.1 (1.0)
run memory Memory usage after adding 1000 rows.	7.6 ± 0.0 (1.0)	10.5 ± 0.0 (1.5)	7.2 ± 0.0 (1.0)
update eatch 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	8.5 ± 0.0 (1.2)	10.6 ± 0.0 (1.5)	7.3 ± 0.0 (1.0)
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	9.0 ± 0.0 (1.2)	10.8 ± 0.0 (1.5)	7.3 ± 0.0 (1.0)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	4.7 ± 0.0 (1.2)	7.1 ± 0.0 (1.9)	3.8 ± 0.0 (1.0)

(b) Memory allocation in MBs +/- standard deviation

Figure 3.6: Performance test of front-end frameworks [Source: Krause[6]]

Templating and Syntax

When comparing Angular, React, and Vue, they also a differ regarding the way of dealing with HTML. In React there is no real HTML, it is called JSX. Even if it looks the same it is Javascript. It is just another way to write a *React.createElement* declaration. The JSX-compiler transforms the JSX code back to React.createElement and executes the call. So it is much more comfortable for the developer than writing native Javascript for a view representation. [Horton and Vice, 2016]

JSX needs getting used to if a developer reads a code line like that in listing 3.2 and it is not HTML. There are self closing divs and tags, for example, which are not supported by HTML. This means there is a also difference in they way it is coded. JSX is placed inside of a React component, to be exact, inside of the render function or all functions which return JSX. There is no separation of concern in terms of Javascript and HTML.

```
const element = <h1>Hello, world!</h1>;
```

Listing 3.2: React JSX example

Angular has a different approach on that. There is the possibility to have HTML and Typescript code in the same file but, according to the style guide of Angular for more complex views, a separate HTML file should be created. Conditional logic can be included in the HTML templates, with the so called structural or attribute directives. An example for that is the following code snippet in listing 3.3.

```
<div *ngIf="show">Angular template</div>
```

Listing 3.3: Angular template example

The structural directive *ngIf is used for conditional rendering. In that case the div with the text Angular template is only rendered in the DOM if the variable show is evaluated to *true*. There are other structural directives such

[6]http://www.stefankrause.net/js-frameworks-benchmark6/webdriver-ts-results/table.html, accessed 19.04.2018

as *ngFor, which loops trough an array. If a developer wants to do the same in React, has to use, for example, the Javascript built-in function map over an array. As there is no inner framework support for that, it has to be done in native Javascript. Vue also has some kind of structural directives, if a developer wants to use conditional logic in a template it looks as in listing 3.4, for completeness the same for React has to be done like in listing 3.5.

```
<span v-if="show">Vue template</span>
```

Listing 3.4: Vue template example

```
<div>
  { show && <span>React template</span> }
</div>
```

Listing 3.5: React template example

In Vue, the template, the Javascript code and even the CSS code are placed in the same file, which are referred to as single file components, but it is still possible to split it in separate files. Angular and Vue have a official style guide, which is very detailed and a huge benefit for developers. Since React is a library, there is no official style guide available, and if more developers work on the same project, there is a chance of inconsistency in code style besides that there is no official best practice.

State management

All candidates supports a state management system. Initially it was developed by Dan Abramov for React, and called Redux, it is considered as the best pattern. for this approach. There is a light weight alternative, which is called MobX but still Redux is the most used pattern. For Vue there is Vuex and for Angular there is ngrx. Both of them are based on the Redux implementation. The motivation of a state management is described as follows:

> ”As the requirements for JavaScript single-page applications have become increasingly complicated, our code must manage more

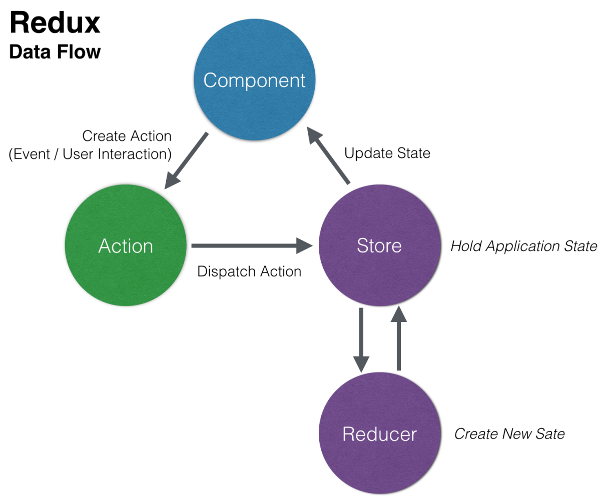


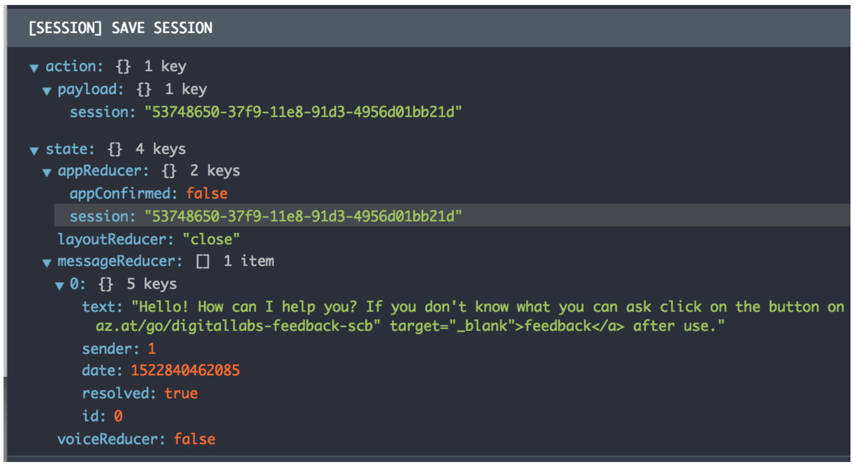
Figure 3.7: Redux state management [Eschweiler, 2017]

> state than ever before. This state can include server responses and cached data, as well as locally created data that has not yet been persisted to the server. UI state is also increasing in complexity, as we need to manage active routes, selected tabs, spinners, pagination controls, and so on." [Redux, 2018]

Figure 3.7 shows the basic concept of how Redux works. It consists of simply three parts: The actions, the reducers, and the store. There can be a additional middleware, but in general those are the core parts.

The store is the single source of truth in an application. In this application, for example, all messages of the user and the bot will be stored there. To save data to the store, an unidirectional flow has to be passed. First an action has to be dispatched, in that case an action could be *storeMessage*.

The reducers are listening to action types and if a reducer subscribed to an action *storeMessage* the reducer function for that action is called. Those are pure functions and this is the only way to manipulate the store. So after the reducer function executes it returns a new state object. A store is basically an object, where the properties are states. After a reducer executes



```
[SESSION] SAVE SESSION

▼ action: {} 1 key
  ▼ payload: {} 1 key
      session: "53748650-37f9-11e8-91d3-4956d01bb21d"

▼ state: {} 4 keys
  ▼ appReducer: {} 2 keys
      appConfirmed: false
      session: "53748650-37f9-11e8-91d3-4956d01bb21d"
    layoutReducer: "close"
  ▼ messageReducer: [] 1 item
    ▼ 0: {} 5 keys
        text: "Hello! How can I help you? If you don't know what you can ask click on the button on
          az.at/go/digitallabs-feedback-scb" target="_blank">feedback</a> after use."
        sender: 1
        date: 1522840462085
        resolved: true
        id: 0
    voiceReducer: false
```

Figure 3.8: Example of a store

its function, a state object is stored to the store, in other words the property message in that case gets updated.

Figure 3.8 illustrates how a store looks like. It shows the store with the states of the reducers so called sub-slices. So there is the *messageReducer*, the *appReducer*, and the *voiceReducer*. The action *storeMessage*, triggers a function of the *messageReducer* and manipulates the state of this sub-slice. After this update the store informs every component, which are connected to the store, about an update and the ui will update. This is the unidirectional flow of a state management system. Vuex and ngrx are working in the same way but are slightly customized for their frameworks. For every middleware for Redux there is a similar one for the other two. Redux Sagas are Effects for ngrx, for example. There is no downside on the other implementation in comparison to the initial Redux.

Decision

	Angular	**React**	**Vue**
Publisher	Google	Facebook	Vue Technology
Programming Language	Typescript	Javascript	Javascript
Componend based	Yes	Yes	Yes
State management	ngrx	Redux	Vuex
CLI	angular-cli	-	vue-cli
Integrated router	angular/router	Only external	Vue-router
CSS modules	Yes	Has to be configured manually	Yes
Separate HTML/JS	Yes	No	Yes
Official styleguide	Yes	No	Yes

Table 3.1: Comparison of front-end frameworks

Conclusions

To summarize the analysis, table 3.1 shows the differences of the evaluated frameworks. For TU Graz Searchchatbot Angular was chosen because it is the most stable and best maintainable framework. It has an official styleguide, which makes it pretty easy to hand the code over to another developer. Every developer who knows Angular, has immediately an idea of the application because of the given structure. Even if the developer is not familiar with the framework after reading the core concept and the styleguide, it should be straight forward. The supported ngrx state management solution is, as already mentioned, the same as Redux so there is no downside on that.

Angular is based on Typescript, so the whole framework was developed in Typescript. This means that there will be no obstacles with the availability of types also for third party libraries. If a developer wants to use React or Vue with Typescript, there is still the problem of the types support of third party libraries. The importance of Typescript can be explained by the fact that type safety reduces the opportunity of errors, as if the Typescript-Compiler fails the developer can be sure there is a problem that needs to be addressed. This means that the developer has increased stability in the code produced.

The tooling with Angular, is better when compared to React. With angular-cli the developer can spin up an application in a few seconds. It is worth mentioning that also Vue has a great Command Line Interface (CLI) solution. With React the developer has to invest more time on the setting up process. Although, there are existing projects like *create-react-app* it is not the same, because its not owned by React itself and its not a CLI solution. Whenever there is an update of React the publisher of create-react-app has to adapt it otherwise it may not work anymore. Due to that having tooling like that integrated into the framework is the better approach.[Filipova, 2016]

Another fact, which is a benefit for Angular is that Google is the owner of this framework, Vue was published by Evan You. Although, there is an increasing Vue community, with Google there is a big player on board. Also the former version of Angular, AngularJS is still supported with updates by Google. In terms of support this is a huge advantage.

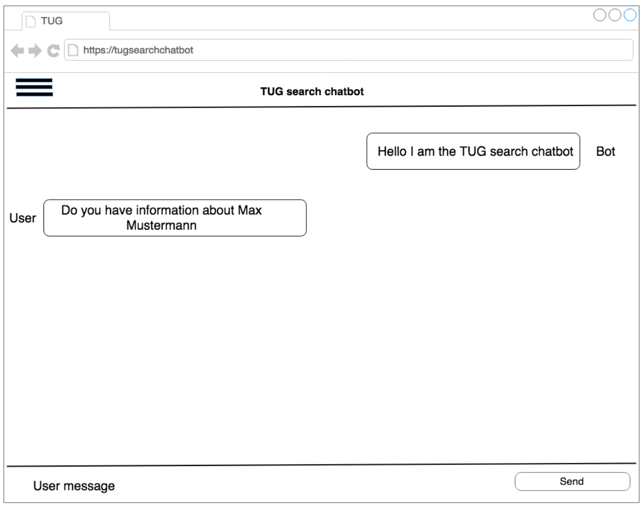

TUG
https://tugsearchchatbot
TUG search chatbot
Hello I am the TUG search chatbot
Bot
User
Do you have information about Max Mustermann
User message
Send

Figure 3.9: Mockup of the chat interface

3.4.2 Design

Based on the requirements, a message interface had to be designed. It should also fit into the corporate identity of the already existing TU Graz website, which means its colour schema should rely on it. In terms of user experience, people should immediately know what the TU Graz Searchchatbot is able to do. In consequence of that, a sidebar with examples was considered to achieve it, since this interface is not restricted in comparison to a standard messenger application. Furthermore, a simple standard chat stream with an input possibility was also designed. Figure 3.9 shows a mockup, built for getting a solid concept before the implementation stage, on how the interface would look like. The interface had to be fully responsive, so it can be easily used in mobile devices. To provide a more complete experience to the users, the messages types shown were analysed. The aim was to understand if there would be the need to style them different styling. The following message types were considered:

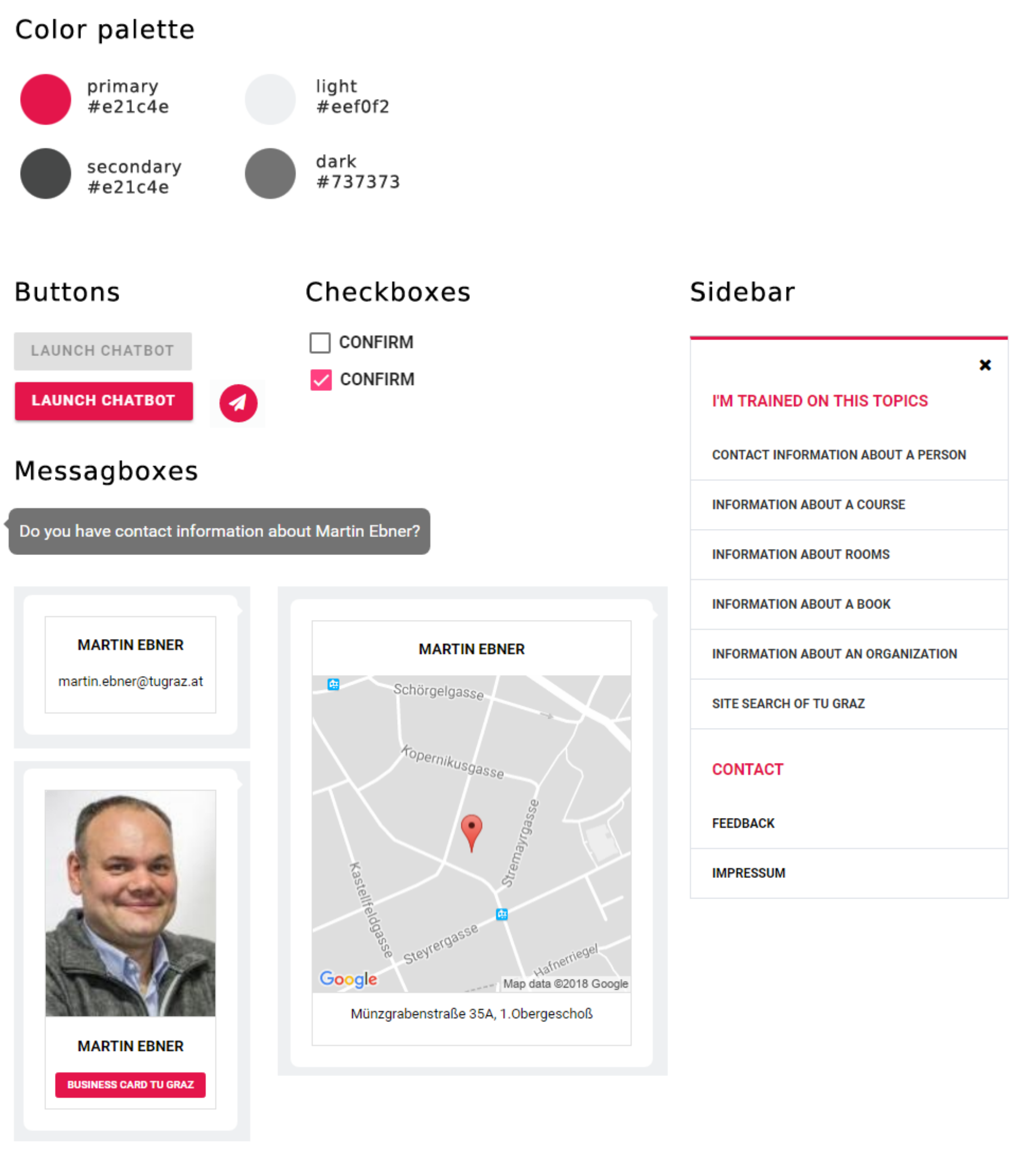
- Simple text message
- Message with google maps integration
- Message with a call to action button
- Message with a user image

Another challenge in a conversational design is how to deal with multiple results. For example, if the user asks for a person and there can be several persons with the same name. If the result exceeds the amount of three, it can not be displayed as a message anymore. Due to that, a modal with the search results should appear with the possibility to show all results and to filter them. In terms of data protection and the use of a third party NLU platform the user has to confirm that he is aware of that. As a result of this there has to be an initial screen which blocks the application until the user confirms.

After considering all use cases a styleguide for the TU Graz Searchchatbot had to be made. A style guide is a guideline for all design elements of an application. This includes the colour schema as well as elements like buttons, modals, or a navigation. Also the typography is part of it, in case of the TU Graz Searchchatbot it is not that important since there are no content pages. The font which was used for this application is Roboto. This font is part of the google fonts library and was published under apache-license[7], which means it is free to use.

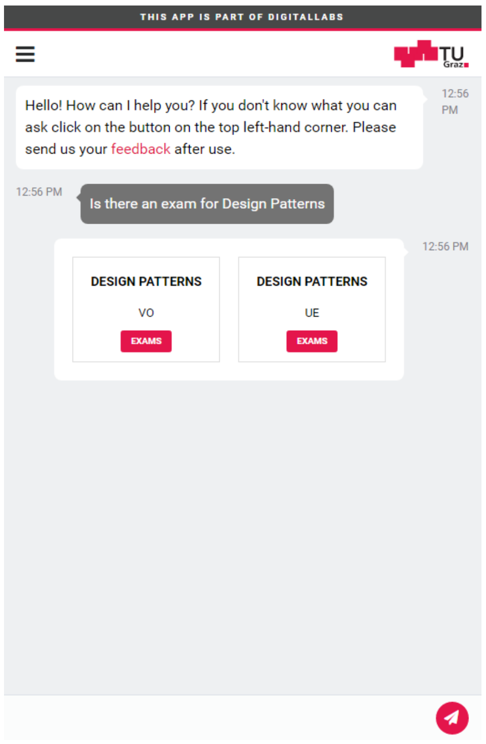
In general a style guide specification has to be done before the development starts because every functional component should rely on the designed element to avoid inconsistencies. Figure 3.10 shows the style guide for the TU Graz Searchchatbot. The primary colour represents the brand colour and it is used for active buttons, links and headlines. The secondary colour is used for other elements like the message boxes, borders, or additional elements. Light and dark are based on the secondary colour to have gradations of it.

[7]https://www.apache.org/licenses/LICENSE-2.0, accessed 25.04.2018



Do you have contact information about Martin Ebner?

Figure 3.10: Styleguide

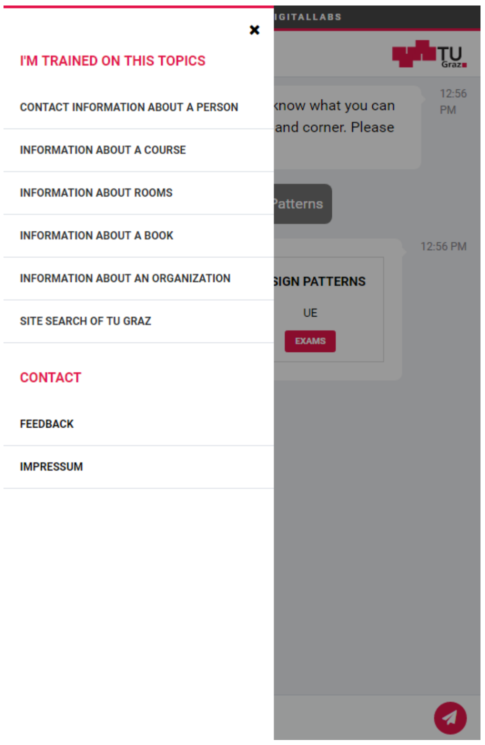


Hello! How can I help you? If you don't know what you can ask click on the button on the top left-hand corner. Please send us your feedback after use.
12:56 PM

12:56 PM
Is there an exam for Design Patterns

12:56 PM

(a) Chatbot interface design



(b) Chatbot interface design with navigation

Figure 3.11: Chatbot interface

After combining all elements of the styleguide the design in figure 3.11 was created. The intention for that design was to keep the simplicity, so it is pretty close to a standard messenger application. The initial bot message points out, that there is a sidebar with basic search examples. Figure 3.11(b) shows the sidebar with trained examples for the bot. It should be a help for the user, to learn how to use the chatbot. If a user clicks an example the sidebar closes and the example text is inserted in the message stream. Right after the user experience was defined and the design phase was finished the implementation could be done.

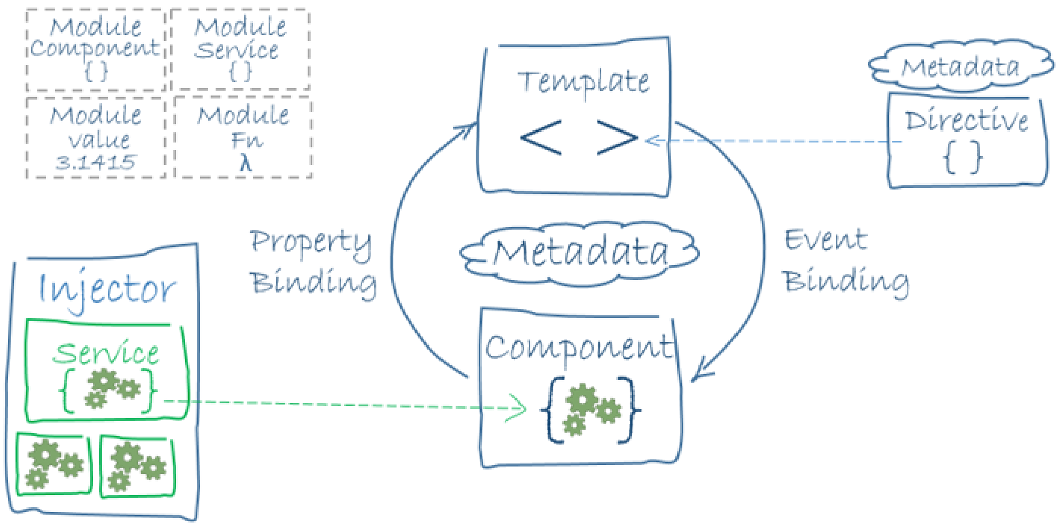


Figure 3.12: Architecture of Angular [Source: Angular[8]]

3.4.3 Implementation

The javascript framework for the TU Graz Searchchatbot is Angular as discussed in the section 3.4.1. In terms of tooling Angular-cli[9] is used and with it also webpack[10]. Webpack is a module bundler, which bundles the CSS and Javascript files, generates chunks for lazy loading, supports hot module replacment, provides a development server, and many more. The initial setup of the project is done via the cli with the command **ng new**. The webpack setup, the folder structure and the root module of the application are created after executing this command. In case of this application the webpack configuration has to be modified. If a developer uses angular-cli, it is not possible to modify the webpack configuration except if it is ejected. After ejecting the configuration the necessary settings for the proxy could be made. The disadvantage of ejecting a webpack config is, that the developer can not use the **ng** commands anymore.[Vepsäläinen, 2016]

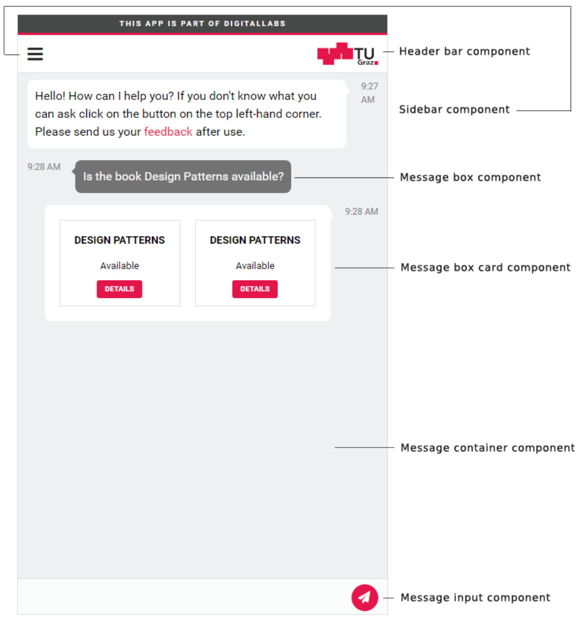
Angular consists of modules, components, services, pipes, and directives.

[8]https://angular.io/guide/architecture, accessed 19.04.2018
[9]https://cli.angular.io/, accessed 19.04.2018
[10]https://webpack.js.org/, accessed 19.04.2018

Figure 3.12 shows the basic architecture concept of the framework. For this application there will be two modules, the already created app module which is the root module, and a shared module. The shared module will be integrated into the app module. It contains all shared components and pipes. All other parts will be done in components which are belonging to the app module. For the interaction with the application interface, ngrx together with ngrx effects will be used. There will be some other third party modules, but they are not in responsibility of the creator of this thesis. The project structure looks as follows:

- App module
  - basic overlay component
  - header bar component
  - message container component
    - message box
    - message simple card
    - message card
    - message maps
    - message overlay
    - message container service
  - message input component
  - sidebar component
- Shared module
  - loader component
  - pagination pipe
  - transform to link pipe
- Store
  - actions
  - effects
  - reducers

Every component has its own responsibility and is injected into its module. Store is only a folder to structure responsibilities, the store module itself is imported from the already mentioned ngrx package. Section3.4.1 explains how the redux pattern works. The message container component has sub-components. Those are responsible for the message boxes in the styleguide,



Hello! How can I help you? If you don't know what you can ask click on the button on the top left-hand corner. Please send us your feedback after use.

9:27 AM

9:28 AM

Is the book Design Patterns available?

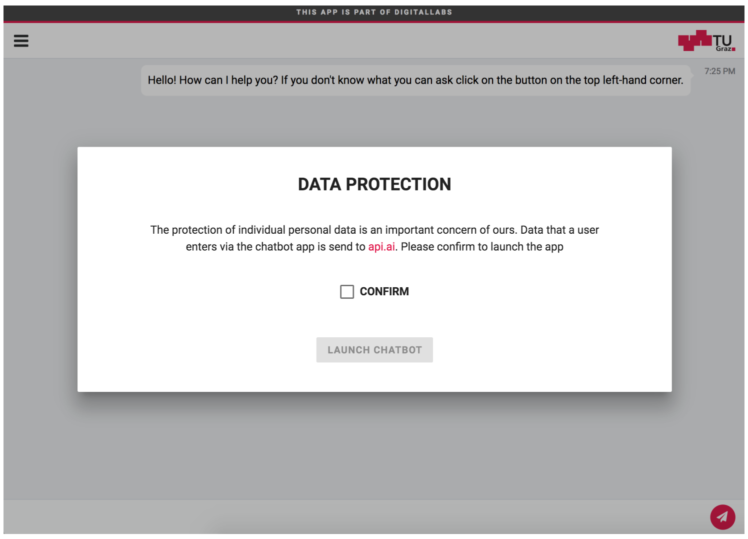
9:28 AM

Figure 3.13: Application components

being an example of that the the message box with google maps integration. Every type of a message box is handled with a separate component. Figure 3.13 illustrates the dependency between the application components and the user interface.

The application works in the following way: After the start up the app module is loaded, this is the entry point of the application. Within this module a root component has to be defined via the bootstrap property. In nearly every case this is the app component. The app component has to lock the application until the user confirms the data protection information. For this purpose, a modal is shown and internally the application is blocked until the confirm application flag is set to *true*. This flag is stored within the ngrx data store. Figure 3.14 shows the initial modal. Simultaneously, the app.component triggers the first bot message *Hello! How can I help you? If you don't know what you can ask click on the button on the top left-hand corner.* Additionally, an API call is sent to receive a valid session.

If the user confirms the data protection information, messages can be sent



7:25 PM

Hello! How can I help you? If you don't know what you can ask click on the button on the top left-hand corner.

DATA PROTECTION

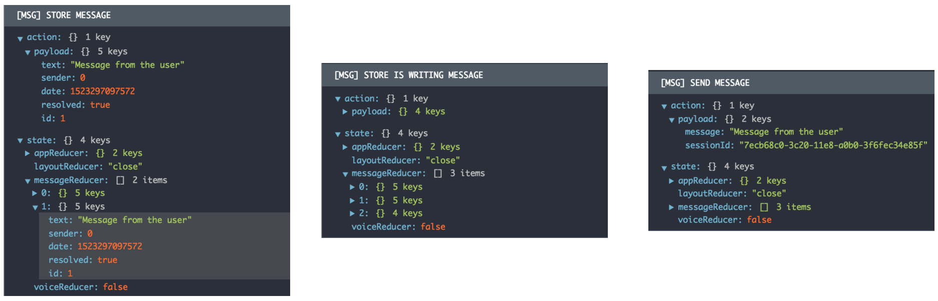
The protection of individual personal data is an important concern of ours. Data that a user enters via the chatbot app is send to api.ai. Please confirm to launch the app

Figure 3.14: Data protection modal

via the message input component. When a message is submitted, three different actions are dispatched. First, the message the user entered is stored via the StoreMessage action, the second is a *Bot is typing* message and it is stored via the action StoreMessageIsWriting and lastly the message is sent to the API via the action SendMessage. The actions StoreMessage and StoreMessageIsWriting are updating an array in the data store; to be more exact, the reducer slice messages which is an array of type messages. Figure 3.15 shows the described procedure. Analysing it, it can be seen that the message reducer gets updated, after dispatching an action with a corresponding payload. Following actions can be dispatched in terms of the message reducer:

- StoreMessage - stores the message of a user
- StoreIsWritingMessage - stores the *Bot is typing* message
- UpdateMessage - updates a message
- DeleteMessage - deletes a message
- SendMessage - sends the entered message to the api

The *Bot is typing message* is stored in the message reducer as long as the API has not responded. After the response arrives, this message gets updated



```
[MSG] STORE MESSAGE
▼ action: {} 1 key
  ▼ payload: {} 5 keys
      text: "Message from the user"
      sender: 0
      date: 1523297097572
      resolved: true
      id: 1

▼ state: {} 4 keys
  ▶ appReducer: {} 2 keys
    layoutReducer: "close"
  ▼ messageReducer: [] 2 items
    ▶ 0: {} 5 keys
    ▼ 1: {} 5 keys
        text: "Message from the user"
        sender: 0
        date: 1523297097572
        resolved: true
        id: 1
    voiceReducer: false
```

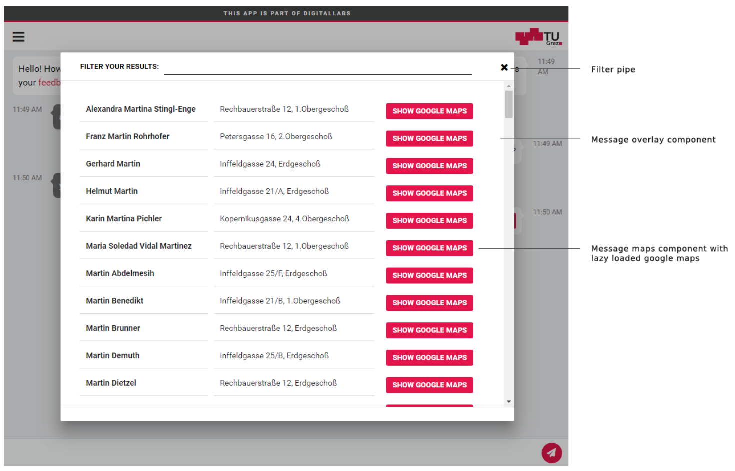
```
[MSG] STORE IS WRITING MESSAGE
▼ action: {} 1 key
  ▶ payload: {} 4 keys

▼ state: {} 4 keys
  ▶ appReducer: {} 2 keys
    layoutReducer: "close"
  ▼ messageReducer: [] 3 items
    ▶ 0: {} 5 keys
    ▶ 1: {} 5 keys
    ▶ 2: {} 4 keys
    voiceReducer: false
```

```
[MSG] SEND MESSAGE
▼ action: {} 1 key
  ▼ payload: {} 2 keys
      message: "Message from the user"
      sessionId: "7ecb68c0-3c20-11e8-a0b0-3f6fec34e85f"

▼ state: {} 4 keys
  ▶ appReducer: {} 2 keys
    layoutReducer: "close"
  ▶ messageReducer: [] 3 items
    voiceReducer: false
```

Figure 3.15: Message flow


THIS APP IS PART OF DIGITALLABS
TU Graz
Hello! How
your feedb
11:49 AM
11:50 AM
11:49 AM
11:49 AM
11:50 AM
Filter pipe
Message overlay component
Message maps component with lazy loaded google maps

FILTER YOUR RESULTS:

Alexandra Martina Stingl-Enge	Rechbauerstraße 12, 1.Obergeschoß	SHOW GOOGLE MAPS
Franz Martin Rohrhofer	Petersgasse 16, 2.Obergeschoß	SHOW GOOGLE MAPS
Gerhard Martin	Inffeldgasse 24, Erdgeschoß	SHOW GOOGLE MAPS
Helmut Martin	Inffeldgasse 21/A, Erdgeschoß	SHOW GOOGLE MAPS
Karin Martina Pichler	Kopernikusgasse 24, 4.Obergeschoß	SHOW GOOGLE MAPS
Maria Soledad Vidal Martinez	Rechbauerstraße 12, 1.Obergeschoß	SHOW GOOGLE MAPS
Martin Abdelmesih	Inffeldgasse 25/F, Erdgeschoß	SHOW GOOGLE MAPS
Martin Benedikt	Inffeldgasse 21/B, 1.Obergeschoß	SHOW GOOGLE MAPS
Martin Brunner	Rechbauerstraße 12, Erdgeschoß	SHOW GOOGLE MAPS
Martin Demuth	Inffeldgasse 25/B, Erdgeschoß	SHOW GOOGLE MAPS
Martin Dietzel	Rechbauerstraße 12, Erdgeschoß	SHOW GOOGLE MAPS

Figure 3.16: Message overlay component with filter

with the new message text. The updating process is triggered by the ngrx effects middleware. The application shows the entered and stored messages in the following with following procedure: Entering and sending the message is the active part inside of the message input component, the passive part is done by the message container, it displays the message array. This component is connected to the data store and whenever there is an update in the message reducer, the new data are shown. In essence, it loops over the message array and every entry a message box component will be then rendered. Inside of the message box component, its type is evaluated, in the case it is a simple text message, so it can be shown. If it is of another type, it works the same way, for all the other message box types, with the only difference that the specific type component is rendered, as mentioned before.

As can be seen with more granularity in the section 3.5, the message model has a property for multiple results and also an indicator for maximum results. If the multiple results array length is bigger then the allowed maximum, then the message overlay component is rendered. Within this overlay the user has the opportunity to filter the given results via the message overlay pipe. A pipe is special component in Angular which enables a transformation of a given variable, in that case the message array. It filters the message array and returns a new array with the expected values. Figure 3.16 illustrates the message overlay component.

The google maps integration in the message maps component is done via static maps[11]. There is also the possibility to lazy load the image, because if the message map component is part of list in the overlay it would be to much traffic to load the map for example for fifty entries. Therefore the lazy loading property is implemented in this component.

Every cross component interaction is done via the ngrx data store, the same happens for user interface actions like opening or closing the sidebar. For the purpose of completeness, a list of the application and layout reducers with their corresponding actions is provided.

[11]https://developers.google.com/maps/documentation/static-maps/, accessed 19.04.2018

- appReducer
  - ReceiveSession - getting the session from the API
  - SaveSession - saving the session in the data store
- layoutReducer
  - OpenSidebar - setting the state value to *true*
  - CloseSidebar - setting the state value to *false*

To get a version for production, the implemented code has to be bundled. To do so, in the command line **npm run build** had to be executed. After executing that command, a productive version is built under the *dist* folder. For the digitallabs version, there is another build command. The reason for that is to support another base url and a different Google API key for the static maps integration. To build a productive version for digitallabs the command **npm run build:digilabs** has to be executed.

3.5 Back-end

This section covers the evaluation process of the Node.js framework for the developed application. Therefore, two widely used frameworks are compared and evaluated. After that the architecture of the back-end is illustrated, as well as the underlying implementation.

3.5.1 Evaluation of Node.js Frameworks

General comparison

With the rise of the importance of Javascript Node.js frameworks are widely used and they are a serious alternative to PHP frameworks like Symfony or Laravel. In this field, there are several frameworks but there is one which stands out in the way is accepted by the community. This Node.js framework is called Express.JS, or simply Express, and was published in 2010. In figure

[12]http://www.npmtrends.com/sails-vs-express-vs-hapi-vs-koa, accessed 19.04.2018

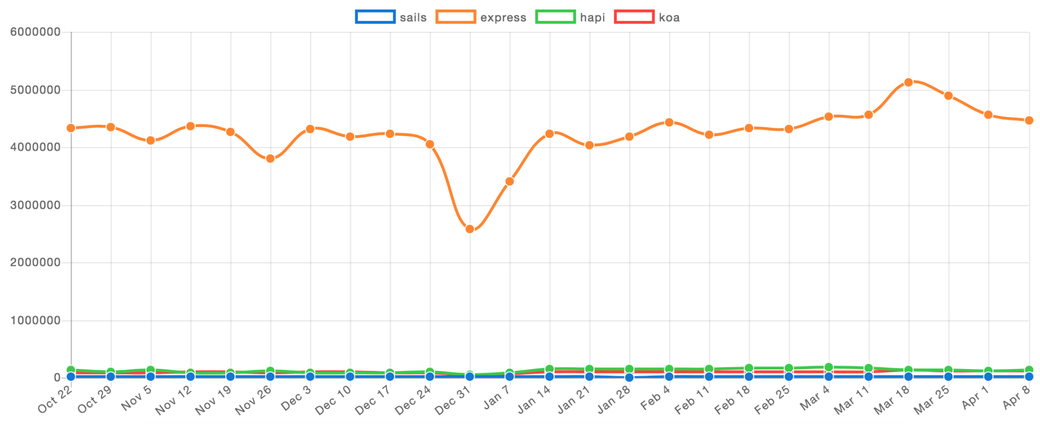


Figure 3.17: Downloads of Node.js frameworks in the last six months. [Source: npmtrends[12]]

3.17 the current download numbers can be seen, and the most popular in the previous six months is also Express. There are some other good Node.js frameworks, but one very interesting alternative to Express is hapi[13]. Hapi was built to mitigate issues faced while using Express. Due to that fact, those two frameworks will be compared in this section.

Before comparing Express and hapi lets take a quick look what Node.js is in general. Node.js is a server-side javascript platform on the base of Google Chrome V8 engine. It is a huge advantage for Javascript developers to realize full stack application because there is no context switch needed. The following section describes the basic concept of Node.js.[Hezbullah Shah, 2017]

The idea of using Javascript on server-side might seem odd for developers who are not familiar with that approach, but there are advantages in comparison to pure server-side programming languages like PHP, Ruby or Java. Node is fast and it is developed by a full-time team in Google, which constantly improving its performance with each release of Chrome. Node is based on V8, therefore it exists a strong relation between Chrome and Node. The biggest difference between Node and other server-side technologies is that it is based on a single-thread approach and the asynchronous coding

[13]https://hapijs.com/, accessed 25.04.2018

style which is a common concept in Javascript on the client side. Other than that Node.js has many other advantages, which are as follows: [Brett, 2016]

- No need for multi-threading, which is always a challenge in application development
- No context switching between client and server side. Possibility of code sharing between server and browser
- Node is bundled with npm currently one of the best package managers. It manages all Javascript application dependencies on server and client side.

[Brett, 2016]

There is to say that there are also alternative package manager out there. Yarn is the most known one which is a widely used as well. With an understanding of what Node.js is and what it does in general the mentioned frameworks can be compared. In essence, Express and hapi have completely different approaches. Express is a lightweight and flexible framework and has a low learning curve, whereas hapi is the opposite, providing a rich feature set. The definition of Express is as follows.

> ”Express.js is a web framework based on the core Node.js http module and Connect components. Those components are called middleware. They are the cornerstone of the framework’s philosophy, which is configuration over convention.” [Mardan, 2014]

The definition of hapi already indicates the enterprise approach with a lot of functionality.

> ”hapi.js (commonly referred to as hapi) stands for HTTP API. It is a rich framework for building applications and services. It was originally designed for the rapid development of RESTful API services using JavaScript, but has since grown to be a full web application framework with out-of-the-box features for templating, input validation, authentication, caching, and more recently,

> support for real-time applications with web socket support." [Brett, 2016]

Due to the contribution to the large and active community of Express, there are a lot of third party libraries available. For that reason, Express can be extended in a way it fulfils the needs of the project. As an example, if an application needs authentication, the integration with a third party library is needed. By contrast, Hapi already provides authentication, which then only needs to be configured. In general, hapi is more configuration centric to reduce code and hence the learning curve is steep.

Splitting an application into several parts is not done very easily in Express. If a larger team is working on the project there is the change that the application will break, because there is no clear indication on how to properly decouple code. Hapi has a plugin system for that, so there is a very clear way to split responsibilities. Another fact worth mentioning is, that hapi has a test coverage of a hundred percent, and it is stress tested under a realistic production atmosphere. It was built by the team of Walmart, which has the particular case of needing be stable on the popular black friday sale, so security and stability have a high priority when developing the framework, as the economic interest if the company can not be jeopardized.

Generating an API

By developing an API the routes have to be defined. Hapi and Express have a different approach on how to implement them. Listing 3.6 shows how a hello world route can be implemented with Hapi. The same example is done with Express in listing 3.7. With this simple example it is already possible to observe the differences between both frameworks. Hapi provides a server.route function, which takes a single configuration object. With Express, the HTTP verb *get* already indicates the method to be used.

```
server.route({
 method: 'GET',
 path: '/',
 handler: function (request, reply) {
 return reply('Hello World\n');
 }
});
```

Listing 3.6: Hapi hello world example

```
app.get('/', function(request, response){
 request.send('Hello World');
});
```

Listing 3.7: Express hello world example

Hapi, as already mentioned, is a framework which has a lot of features out of the box. If there is the need to read the body of an incoming message, which is a common use case, then it can be done in Hapi without any additional work. The same task can not be done in Express without an external library, in that case the body-parser[14]. Furthermore, the validation is part of the Hapi environment, it is called Joi. Listing 3.8 shows an example with a parameter validation.

[14]https://www.npmjs.org/package/body-parser, accessed 19.04.2018

```
server.route({
  method: 'GET',
  path: '/hello/{name}',
  config: {
    description: 'Return an object with hello message',
    validate: { params: { name: Joi.string().min(3).required()
   }  },
    pre: [],
    handler: function (request, reply) {
      const name = request.params.name;
      return reply({ message: `Hello ${name}` });
    },
    cache: { expiresIn: 3600000 }
  }
});
```

Listing 3.8: Hapi parameter validation, [Brett, 2016]

Listing 3.9 shows an example how to use the body-parser in Express to provide the same functionality as hapi.

```
const bodyParser = require('body-parser');
...
const app = express();
...
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
...
app.post('/hello', function(request, response){
 request.send('Hello' + request.body.name);
});
```

Listing 3.9: Express body-parser example

Decision

After evaluation Express and Hapi a decision could be made. One one hand Hapi is the more powerful and convenient framework, but on the other Express attracts attention for its simplicity and minimalist approach. For complex enterprise applications, Hapi is the better framework. For this thesis, however, Express is used, given the there will be only two endpoints to communicate with the client. Hapi's feature set would be a bloat of functionality and it is not used for this application.

3.5.2 Implementation

Since there is a strong connection between the Node.js back-end, which acts as a middleware, and the NLU platform dialogflow.ai. To ensure a solid base of work, the setup had to be analysed before the start of the implementations. The structure of the TU Graz Searchchatbot dialogflow project is the base for the back-end implementation.

Setup of dialogflow

There are three main components in dialogflow to build a chatbot. Those are:

- Intents
- Context
- Entities

It is important to understand what they are and what their role is. Referring to the documentation

> ”An intent represents a mapping between what a user says and what action should be taken by your software.” [dialogflow, 2018]

This means that for every search topic there has to be an intent. For every intent sample, sentences have to be added so dialogflow recognizes the desired intent correctly. For this application intents for the following search topics have to be added:

- Contact information - about a person
- Course information - about a course

- Library information - about books
- Organization information - about an organization
- Room information - about a room
- Site search - triggers a site search of the TU Graz website

An intent has the possibility to set an input and output context. A context is a special object witch stores information. For example, if there are follow up questions, data which was already in this context can be accessed, to be more precise, via the context object values can be passed between intents. For the intent content information the context looks as follows:

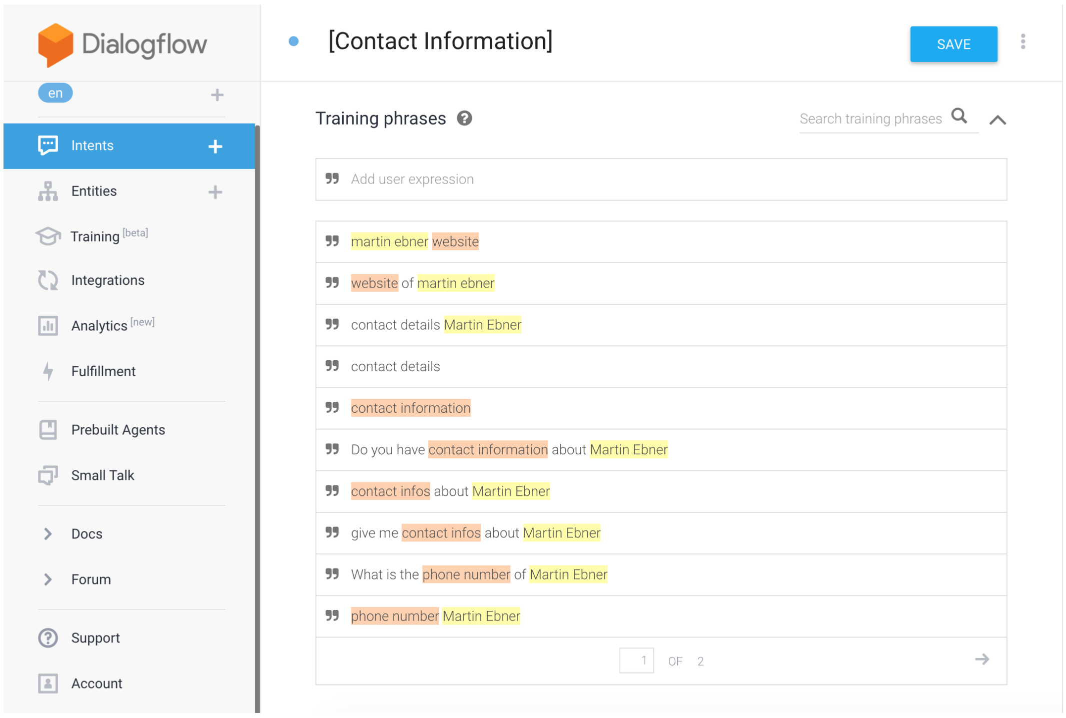
- Input context: none
- Output context: ctx_contact_information

The input context is none in that case because it is an entry point of a chat. The output context is prefixed with *ctx* and defines the shared object for the contact information flow. Next training phrases had to be added which can be seen in figure 3.18. Certain text parts of every training phrase are highlighted. This indicates the corresponding parameters. A parameter belongs to an entity, the main component of dialogflow. For the training phrase *Do you have contact information about Martin Ebner?* there are two parameters with different entities which are:

- contact_type: contact information
- any: Martin Ebner

Since there is no proper name recognition for german names in dialogflow, this parameter is of entity *any*, which means this could be any name. The parameter *contact_type* has a custom entity which is also called *@contact_type*. Figure 3.19 illustrates how an entity looks like. There is a list of references values like *p_tel_office_PA* and their corresponding synonyms, in that case *phone number, phone, number*. The reference value matches the field name of the XML result in 3.1. Finally, an action which has to be performed for this intent has to be defined, for this example it is *act_get_contact_info* is set.

Now, an example query can be made. If a user asks for *Do you have contact information about Martin Ebner?* the dialogflow API responds with the JSON in listing 3.10.



[Contact Information]

Training phrases

Add user expression

martin ebner website

website of martin ebner

contact details Martin Ebner

contact details

contact information

Do you have contact information about Martin Ebner

contact infos about Martin Ebner

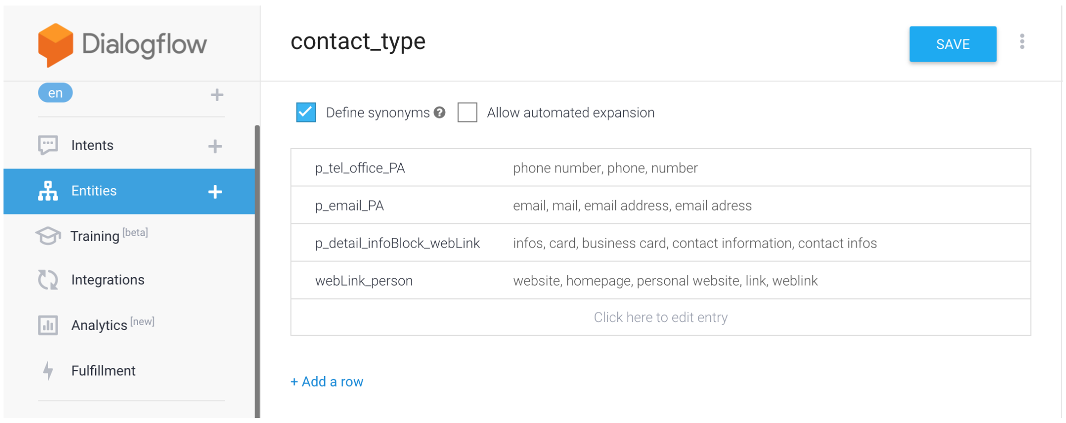
give me contact infos about Martin Ebner

What is the phone number of Martin Ebner

phone number Martin Ebner

1 OF 2

Figure 3.18: Dialogflow intent training phrases



contact_type

[✓] Define synonyms [ ] Allow automated expansion

p_tel_office_PA	phone number, phone, number
p_email_PA	email, mail, email address, email adress
p_detail_infoBlock_webLink	infos, card, business card, contact information, contact infos
webLink_person	website, homepage, personal website, link, weblink

Figure 3.19: Dialogflow example entity

```
{
  "id": "11256efc-1fa6-4ef9-aef0-3d6b03f2e610",
  "timestamp": "2018-04-11T14:42:02.968Z",
  "lang": "en",
  "result": {
    "source": "agent",
    "resolvedQuery": "Do you have contact information about
   Martin Ebner?",
    "action": "act_get_contact_info",
    "actionIncomplete": false,
    "parameters": {
      "any": "Martin Ebner",
      "contact_type": "p_detail_infoBlock_webLink"
    },
    "contexts": [
      {
        "name": "ctx_contact_information",
        "parameters": {
          "contact_type": "p_detail_infoBlock_webLink",
          "contact_type.original": "contact information",
          "any.original": "Martin Ebner",
          "any": "Martin Ebner"
        },
        "lifespan": 5
      }
    ],
    "metadata": {
      "intentId": "c35628dc-974f-4659-a551-b0ebfe4dd150",
      "webhookUsed": "false",
      "webhookForSlotFillingUsed": "false",
      "intentName": "[Contact Information]"
    },
    "fulfillment": {},
    "score": 1
  },
  "status": {
    "code": 200,
    "errorType": "success",
    "webhookTimedOut": false
  },
  "sessionId": "99c33c40-c101-499c-a907-8281e5be1d60"
}
```

Listing 3.10: Dialogflow response

In listing 3.10 all the discussed values can be seen. One interesting property in the JSON response object is *actionIncomplete*. This property signals that the intent is already fulfilled. For example, if the question would be *Do you have contact information?* this property would be *true* because the parameter *@contact_type* is undefined and a follow up question would appear.

If a developer wants to trigger an intent programmatically, this can be done with events. Every intent has the possibility to listen to certain events. For example if a user asks for a phone number of person *martin*, than there will be multiple results. Dialogflow does not know how many results the TU Graz search-proxy found for this query, so the back-end has to fire an event like *evt_specify_contact_information* for that to trigger another intent.

To ensure consistency within the dialogflow project the prefixes in table 3.2 are used for the corresponding types.

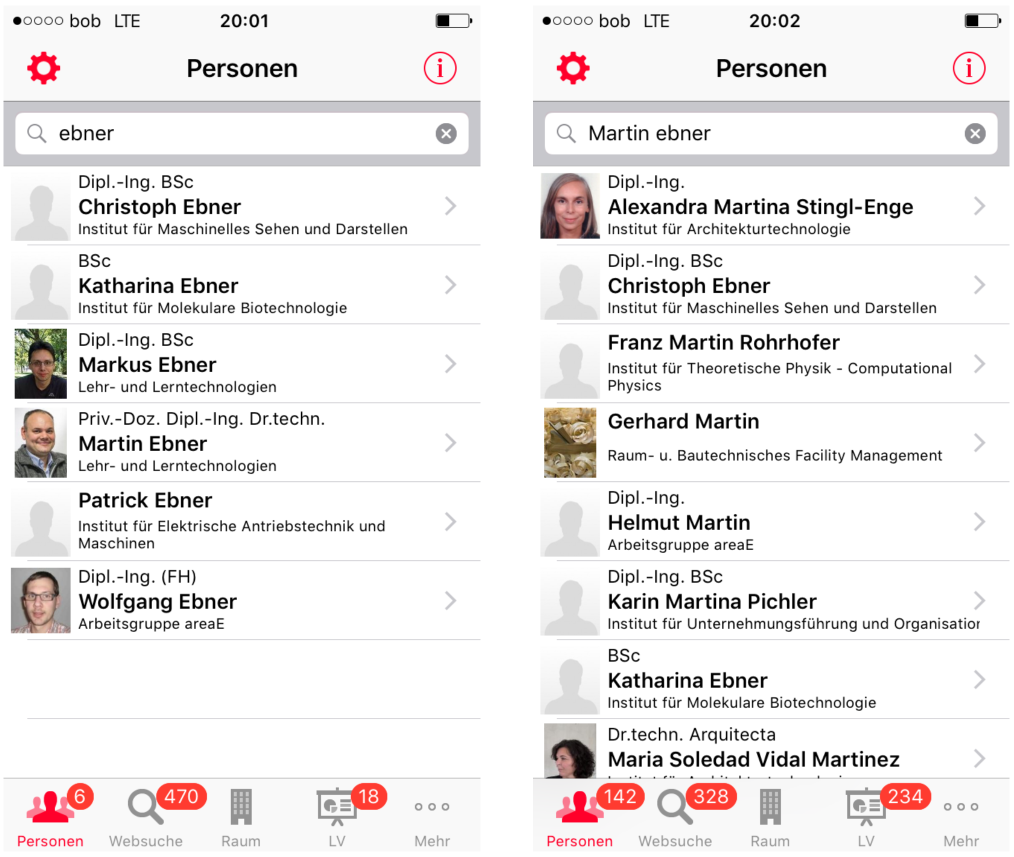
Type	**Prefix**
Action	act
Event	evt
Context	ctx

Table 3.2: Pefixes for dialogflow

Node.js back-end

The Node.js back-end was implemented, with Express. The reason is that it acts more like a middleware and there are only two endpoints for the communication with the client. Nevertheless, there will be business logic in terms of communication with the dialogflow API and the TU Graz search-proxy. The two endpoints are listed in table 3.3.

The back-end should also improve the quality of the search results in comparison to the TU Graz mobile search application. The mobile search application only represents the result of the search-proxy, so there is no business logic in terms of improving the search results. While this is not a big problem in a mobile application it is for a conversational interface.


●○○○○ bob LTE 20:01

Personen

ebner

Dipl.-Ing. BSc
Christoph Ebner
Institut für Maschinelles Sehen und Darstellen

BSc
Katharina Ebner
Institut für Molekulare Biotechnologie

Dipl.-Ing. BSc
Markus Ebner
Lehr- und Lerntechnologien

Priv.-Doz. Dipl.-Ing. Dr.techn.
Martin Ebner
Lehr- und Lerntechnologien

Patrick Ebner
Institut für Elektrische Antriebstechnik und Maschinen

Dipl.-Ing. (FH)
Wolfgang Ebner
Arbeitsgruppe areaE

Personen 6 | Websuche 470 | Raum | LV 18 | Mehr

●○○○○ bob LTE 20:02

Personen

Martin ebner

Dipl.-Ing.
Alexandra Martina Stingl-Enge
Institut für Architekturtechnologie

Dipl.-Ing. BSc
Christoph Ebner
Institut für Maschinelles Sehen und Darstellen

Franz Martin Rohrhofer
Institut für Theoretische Physik - Computational Physics

Gerhard Martin
Raum- u. Bautechnisches Facility Management

Dipl.-Ing.
Helmut Martin
Arbeitsgruppe areaE

Dipl.-Ing. BSc
Karin Martina Pichler
Institut für Unternehmungsführung und Organisatior

BSc
Katharina Ebner
Institut für Molekulare Biotechnologie

Dr.techn. Arquitecta
Maria Soledad Vidal Martinez

Personen 142 | Websuche 328 | Raum | LV 234 | Mehr

(a) Result with last name (b) Result with first name and last name

Figure 3.20: TU Graz mobile search

Endpoint	**Method**	**Parameters**	**Description**
getSession	GET	none	Returns the current session of the user
communication/send	POST	object of type queryMessage	Returns an object of type message

Table 3.3: Back-end endpoints

Figure 3.20 shows the problem. If a user searches for *ebner* it responds with six results. If the user specify the name more exactly with first and last name the application responds with 142 results. The back-end solution will target and solve that issue by doing an additional filtering of the TU Graz search-proxy result.

To ensure the possibility of modular extensibility, a proper folder structure had to be chosen. The idea is to have a main entry point where all future routes can be added. The routes are modularized in terms of their responsibility. In this case the only main route is communication. Under the folder communication there is again a main entry point which handles all actions. According to that the project folder structure looks as follows:

- index.js: *root module to plugin routes*
- routes: *all routes should be placed here*
  - communication: *includes all routes corresponding to communication*
    - actions: *all actions such as queries are placed here*
    - models: *all object types are placed here*
    - course
    - library
    - organizations
    - person
    - room
    - site-search

To provide a proper communication between the client and the back-end object types are defined. Since Typescript is used in the front-end, this types can be specified there as well. Following the types are listed.

```
{
    session: string,
    message: string
}
```

Listing 3.11: Object of type MessageQuery

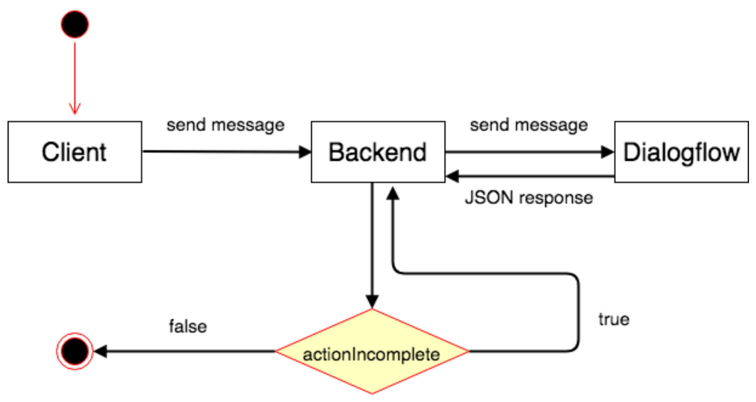


Figure 3.21: Message flow

```
{
    additionalText: string,
    cards: Card[],
    isAdress: boolean,
    isCard: boolean,
    link: string
    multipleResults: array
    text: string
}
```

Listing 3.12: Object of type Message

```
{
    buttonText: string,
    description: string,
    headline: string,
    image: string,
    link: string,
    options: {
        showButton: boolean,
        showDescription: boolean,
        showImage: boolean,
    }
}
```

Listing 3.13: Object of type Card

There are two different flows which can appear during a chat with a user.

On one hand, there can be a message which is fulfilled after one iteration, on the other hand there can be follow up questions. Figure 3.21 shows the different flows in detail. To better understand the base logic behind these flows, the whole back-end process will be described below in detail.

First of all, the node server Express has to be started. There two different starting scripts were to accomplish that with **npm start** for the development mode and **npm prod** for the production version. There is a difference of the used modules for this two running modes. During the development the server should restart on every code change, saving the time to this manually. This is done by nodemon[15] a node module which watches files in the development directory, and restarts the server automatically. While this is helpful for development, it is not needed in the production version. However, the server should restart in case of an error, hence minimising any possible downtime and not compromising the server availability. For that reason, the node module forever[16] was used, which ensures that the server runs continuously.

After setting up the basics the SDK for dialogflow had to be integrated. This is a straight forward process, in which the SDK had to be initialized with an API-key and it was then ready for use.

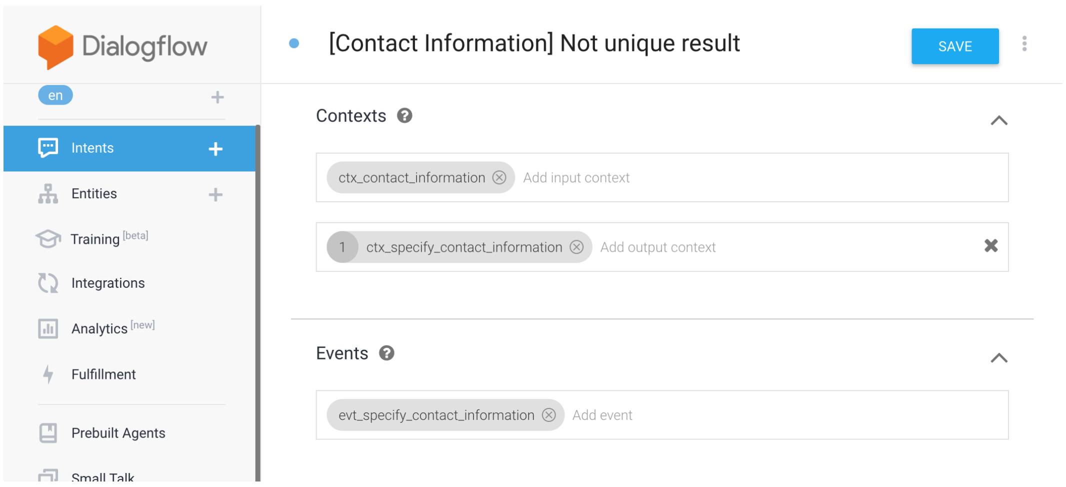
If a user sends a message with the content *Do you have contact information?* the in table 3.3 described endpoint **/communication/send** is triggered. This is the entry point for all message queries. The first thing which is done here, is that the message is forwarded directly to dialogflow which can be seen in listing 3.14.

```
const request = app.textRequest(req.body.message, {
        sessionId: req.body.sessionId
    });
```

Listing 3.14: Dialogflow SDK

Dialogflow responds either with a valid response or an error. Considering the response is valid, the two cases in figure 3.21 were handled. If

[15]https://github.com/remy/nodemon, accessed 19.04.2018
[16]https://github.com/foreverjs/forever, accessed 19.04.2018



[Contact Information] Not unique result

Contexts

ctx_contact_information — Add input context

1 ctx_specify_contact_information — Add output context

Events

evt_specify_contact_information — Add event

Figure 3.22: Dialogflow intent via event

actionIncomplete is *true*, another *textRequest* had to be sent otherwise the action would be evaluated. As discussed above, every intent results in an action, this action is now evaluated. For this example, the action would be **act_get_contact_info** a function call for querying contact information. This is done via a class called TUSearchProxy, illustrated in figure 3.23. The XML response has to be parsed according the *contact_type* of the response of dialogflow. If there is a unique or less than six results, the final response is sent to the client. If this is not the case, then another event is fired. This event is called *evt_specify_contact_information* and triggers an intent in the dialogflow project. Listing 3.15 gives an example how to trigger an event via the dialogflow SDK and figure 3.22 shows the corresponding intent.

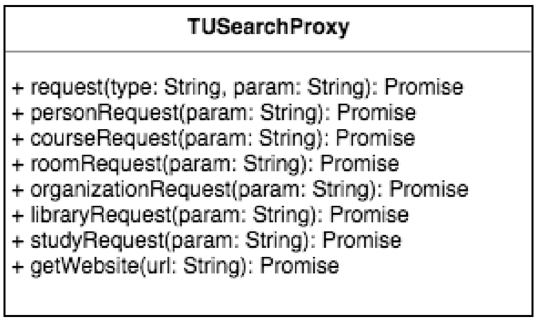

TUSearchProxy
+ request(type: String, param: String): Promise
+ personRequest(param: String): Promise
+ courseRequest(param: String): Promise
+ roomRequest(param: String): Promise
+ organizationRequest(param: String): Promise
+ libraryRequest(param: String): Promise
+ studyRequest(param: String): Promise
+ getWebsite(url: String): Promise

Figure 3.23: TUSearchProxy Class

```
function sendEvent(id, event) {

    const request = app.eventRequest(event, {
        sessionId: id
    });

    request.end();
}
```

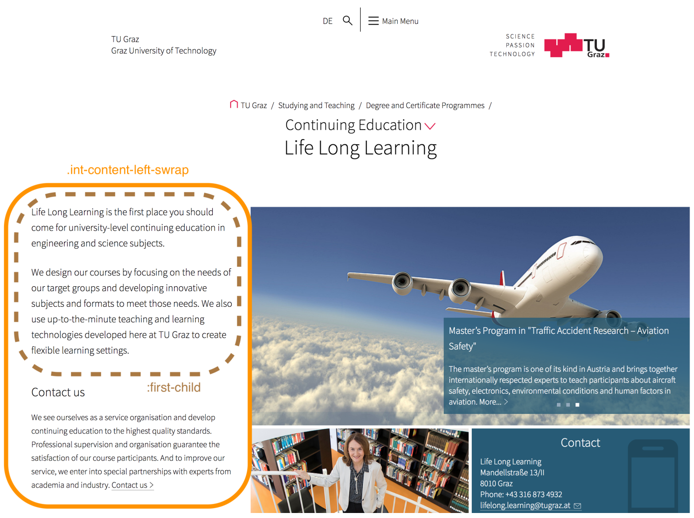
Listing 3.15: Sending an event via the dialogflow SDK

This kind of a loop will appear as long as the conditions described above are the same. After specifying the query and getting an unique result, the message is returned to the client. This works the same for all other action types except the action type *act_get_study_info*. To retrieve information about the study, the search proxy can not be used. Therefore, the site search of the TU Graz website has to be triggered with **http://search.tugraz.at/search?q=term&output=xml&sitesearch=www.tugraz.at/en/studying-and-teaching&ulang=en**. The site search responds with an XML consisting of found subpages of the website corresponding to the search term as listing 3.16 shows. In this example the search term is *life long learning*.

```
...
<RES SN="1" EN="7">
<M>7</M>
<FI/>
<XT/>
<R N="1">
<U>
https://www.tugraz.at/en/studying-and-teaching/degree-and-
    certificate-programmes/continuing-education/life-long-
    learning/
</U>
<UE>
https://www.tugraz.at/en/studying-and-teaching/degree-and-
    certificate-programmes/continuing-education/life-long-
    learning/
</UE>
<T><b>Life Long Learning</b> - TU Graz</T>
<RK>9</RK>
<CRAWLDATE>10. Apr. 2018</CRAWLDATE>
<ENT_SOURCE>T5-J7PFKQMLX9SBC</ENT_SOURCE>
<FS NAME="date" VALUE="2018-04-10"/>
<S>
<b>...</b><b> Life Long Learning</b>. <b>...</b> Contact. <b>
    Life Long Learning</b> Mandellstraße 13/II 8010 Graz<br>
    Phone: +43 316 873 4932 lifelong.<b>learning</b>
    noSpam@tugraz.at Map. <b>...</b>
</S>
<LANG>en</LANG>
<HAS>
<L/>
<C SZ="127k" CID="PlwTzTR_RnEJ" ENC="UTF-8"/>
</HAS>
</R>
...
</RES>
...
```

Listing 3.16: TU site search result

The XML result has to be parsed and the URL of the first result is taken, which is *https://www.tugraz.at/en/studying-and-teaching/degree-and-certificate-programmes/continuing-education/life-long-learning/, accessed 19.04.2018*. To get the desired information, this webpage has to be rendered. To accomplish



TU Graz
Graz University of Technology

TU Graz / Studying and Teaching / Degree and Certificate Programmes /

Continuing Education

Life Long Learning

Life Long Learning is the first place you should come for university-level continuing education in engineering and science subjects.

We design our courses by focusing on the needs of our target groups and developing innovative subjects and formats to meet those needs. We also use up-to-the-minute teaching and learning technologies developed here at TU Graz to create flexible learning settings.

Contact us

We see ourselves as a service organisation and develop continuing education to the highest quality standards. Professional supervision and organisation guarantee the satisfaction of our course participants. And to improve our service, we enter into special partnerships with experts from academia and industry. Contact us >

Master's Program in "Traffic Accident Research – Aviation Safety"

The master's program is one of its kind in Austria and brings together internationally respected experts to teach participants about aircraft safety, electronics, environmental conditions and human factors in aviation. More... >

Contact

Life Long Learning
Mandellstraße 13/II
8010 Graz
Phone: +43 316 873 4932
lifelong.learning@tugraz.at

Figure 3.24: TU Graz content page

that request[17] was used. With this library the HTML of the website can be accessed. To parse the website cheerio[18] was integrated. Cheerio implements a subset of jQuery[19] and makes the parsing of the body pretty easy. Listing 3.17 illustrates how this procedure is done.

```
request(requestUrl, (err, resp, body) => {
    $ = cheerio.load(body);
    let content = $(SELECTOR).text();
    let fallbackContent = $(`${SELECTOR_FALLBACK} div:first-
  child`).text();

    ....

    resolve({ message: message.getMessage(), event: event.
  getEvent() });
});
```

Listing 3.17: Parsing an TU Graz content page

For parsing the content there are two selectors defined, a standard and

[17]https://github.com/request/request, accessed 19.04.2018
[18]https://github.com/cheeriojs/cheerio, accessed 19.04.2018
[19]http://jquery.com, accessed 19.04.2018

fallback selector. Generally, every content page has the same HTML structure of the TU Graz website. Due to that, selectors can be hardcoded. Figure 3.24 illustrates the parsed sections. The extracted text is than transformed to a message and returned to the client.

In summary, there are actions which refer to the TU Graz search proxy and the study information action which is retrieved via the TU website itself. The latter is stable in a long term, because of a hard dependency to CSS classes, but it can be easily adjusted. Further actions can be added easily in the main entry point as discussed.

4 Discussion

In this chapter the feedback about the Searchchatbot is discussed. In reference to it, there will be an analysis of the current situation and possible improvements will be discussed. Since this is a prototype, the feedback is very important to improve things in an early stage and to better make decisions for the future.

4.1 Feedback

To be able to make a statement about the TU Graz Searchchatbot a feedback form was integrated. Since this bot is in the prototype phase the acceptance of it was questioned. The questions asked were the following:

- How satisfied where you with the Searchchatbot?
- Which search concept would you generally prefer in the future?
- Do you think that the application / the Searchchatbot persist in the long term?
- In your opinion, the bot worked best in search category ...
- What did you like about the Searchchatbot?
- What did you not like about the Searchchatbot?
- Do you have any further comments or suggestions for us?

The considered time period was from end of September 2017 to end of February 2018 and twelve people participated in the survey. It has to be considered that in this period some changes regarding the results of the TU Graz search-proxy appeared, which were not adapted in the Searchchatbot implementation. Following, the questions are discussed in detail. It has to be mentioned that the amount of participants is to less to have a meaningful knowledge. For the next feedback iteration there should be a longer feedback phase to gather information, however trends concerning the Searchchatbot can be recognized.

How satisfied where you with the Searchchatbot?

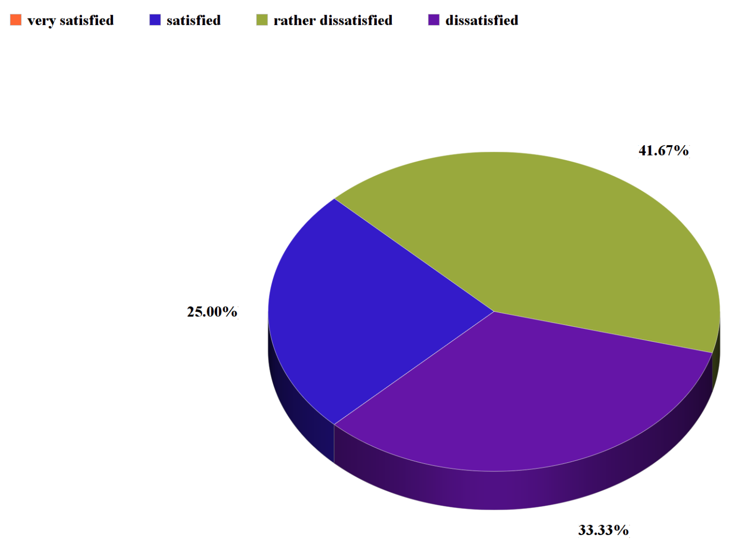


Figure 4.1: Result of "How satisfied where you with the Searchchatbot?"

The rating scale includes the values very satisfied to dissatisfied. A fourth of the people are satisfied, nearly the half kind of and the others were not. So if 66% of the people are in some way satisfied with the bot, which is basically good signal for a prototype. Figure 4.1 represents the result in detail.

Which search concept would you generally prefer in the future?

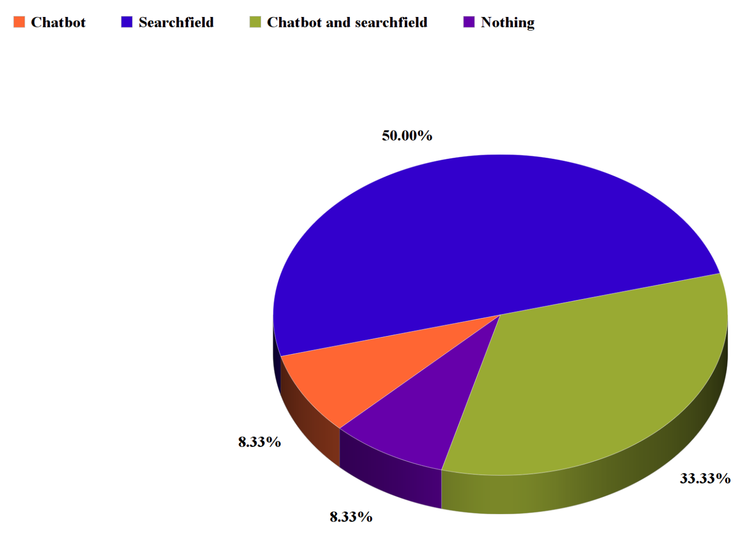


Figure 4.2: Result of "Which search concept would you generally prefer in the future?"

As figure 4.2 shows, half of the users would prefer the standard search field, almost the other half is interested in the bot concept. Also concerning this question, a basic interest is recognizable. It has also to be taken into account that people are not used to chatbots in terms of searching. It may need some time of analysis and optimization to convince people.

Do you think that the application / the Searchchatbot should persist in the long term?

More than a half of the participants thinks that the Searchchatbot should persists as an additional solution as figure 4.3 shows. Again since this is a prototype this is a promising result for the future.

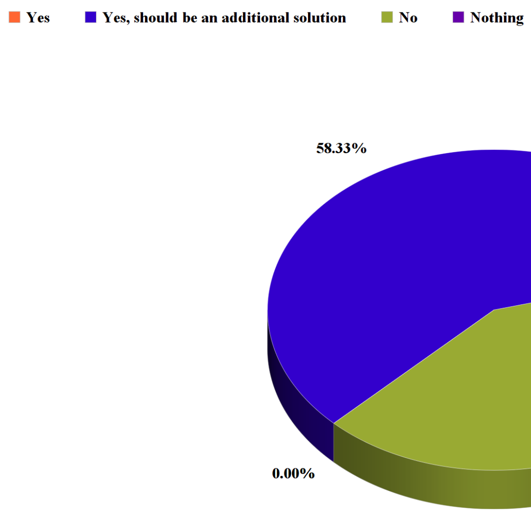


Figure 4.3: Result of "Do you think that the application / the Searchchatbot should persist in the long term?"

In your opinion, the bot worked best in search category ...

As seen in figure 4.4, exactly the half of the participants are in the opinion that the search results where not exactly enough. In a next step this should be solved by serving more or better interfaces to the bot. At the moment only the TU Graz search-proxy is used for fetching data. Providing an API for that with the option of interacting with more meta data would result in a much better response.

What did you like about the Searchchatbot?

Since this is an open question, following is displayed a sample of the answers.

"I like the application Searchchatbot, it is very innovative and

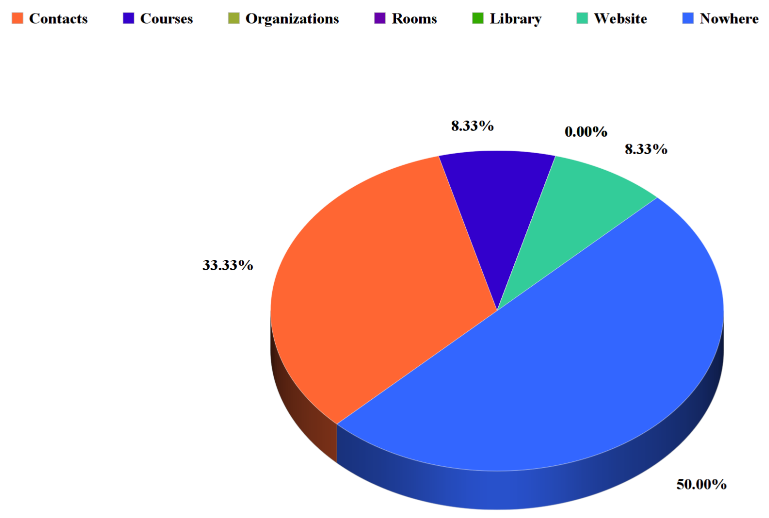


Figure 4.4: Result of "In your opinion, the bot worked best in search category ..."

> invites to a quick search of contact information. I would like to have it with voice recognition and also mobile. It would be nice to connect it with Siri so that a phone number could dialed or a room search could be triggered with Google Maps."
>
> "I like that it is giving direct links to the relevant pages in e.g. the library system or TUGRAZonline but also directly displaying parts of the asked information. When asking for part of a name with many results it was giving me the answer: I have x results matching your search. Do you want to see all?"
>
> "I like the dialog."

It is interesting that especially the design parts which are deviating from the standard messenger are rated positive. For example the dialog with the filter option is not available on a messenger platform. Another interesting aspect is that the bot should also be available on mobile, since it is a responsive version. Hence, a native mobile application would be appreciated by the users.

What did you not like about the Searchchatbot?

Once again, this is an open question, following are examples of answers:

> ″There is no added value to a standard search field. You are limited in advance. Did not find the course Softwaretechnologie SE.″
>
> ″I search for office and room of person x and I received the answer ″Sorry I didn't find information for this course″. The question and the answer does not match. It would be better to have a neutral answer which fits. Would also be better to have some alternative suggestions like ″Did you mean ...″.
>
> I would prefer using the bot if it would be integrated on the OE-pages as a virtual assistant and not only as a standalone version.″
>
> ″It didn't found that much.″
>
> ″It did not quite work. Only when I tried a different way to ask my question did it find the answer. How should I know which sentence to use in order to get a correct answer. It also answered ″right here″ when asked about s.o.'s office. There was no link.″
>
> ″Currently it feels like nothing more than a single access channel to multiple search features of TU Graz systems. If this is all it should be I think a global TU search feature is more reasonable.″

Basically the main problem which can be derived from the answers is again the underlying search resource. In fact there are some problems with finding information if the name for example of a course is not typed correctly or the same as it is stored in the system. To improve that an API should be provided.

Do you have any further comments or suggestions for us?

Some of the answers are displayed below:

> ”Currently it happens very often if I don’t use some predetermined search patterns, that the bot just does not understand. If it is not sure it is better to ask. Did you maybe mean that? I think one should perform multiple searches in all search areas and try to come up with a possible solution more often rather than just say ”I dont know”. The search you do internally should be something like WolframAlpha does when getting a search request but over internal data.”

> ”I think a search bot would be great if you plan to make it more than just one channel to search all TU features. What would be nice is if for instance if I have a search for a person with too many matches it asks me back. ”Do you know any other information about this person?” Then I could answer ”She works in chemistry”, and this way the person can be found. In my opinion only if such multi-round conversations are possible a search bot makes sense instead of search field.”

> ”If you show contact info don’t just show name plus link but also the ”most relevant” info directly in the chat. This saves lots of time in 99% of the cases.”

In general, it can said that more conversational interaction with the bot is desired. Unfortunately, most of those things are not doable at the moment with the search proxy. Again, there is a interest of using a Searchchatbot but it has to be optimized in terms of data resources.

4.2 Evaluation and Improvements

By analysing the results, the Searchchatbot has positive feedback signals of acceptance. There are things to improve, which can be done by the implementation of the Searchchatbot itself, but the most important thing is to provide a better searching API towards the Searchchatbot back-end. To clarify and to point out which things can be fixed with the current implementation, a quick recapitulation follows of what dialogflow is able to do and what is not possible at the moment.

Dialogflow supports an existing application with natural language processing, so it is able to understand intents and parses a sentence accordingly. It kind of provides a chatbot which is simply based on a decision tree. It supports machine learning in terms of given examples, but if a user asks a question and the bot does not understand it, next time the bot will answer exactly the same because there is no artificial intelligence providing changes in that way.

Due to that its flexibility is reduced, which means that the user inputs, especially in the first feedback iteration, has to be logged and analysed, so the search patterns can be adapted to improve the bot usefulness to its users. As described in section 3.5.2, names are parsed as a entity with the type **any**, so there is a chance of misunderstanding. This was opted for implementation, because at the moment there is no support for German. To fix that there has to be a possibility to add names to improve parsing. Double names are also critical at the moment, which can be solved with a pattern adaption.

The biggest benefit in kind of user experience would be more interaction with the bot. To accomplish that, more information resources need to be provided to answer questions in a better way. The TU Graz search-proxy could continue to be used, but as a single information resource it is unsatisfactory for that use case. Providing an API which is dedicated to that purpose would be the best way to achieve it. Since there is a Node.js back-end in use, the response should be an JSON object to improve the processing of the data. This would also improve maintainability of the current system.

5 Summary and outlook

The aim of this thesis was to build a standalone Searchchatbot prototype for the TU Graz which provides the search functionality of the TU Graz search mobile application. Literature was reviewed to give an overview of chatbots in general. Several types were discussed and the rise of the chatbots was explained. Also a comparison of business and customer chatbot approaches was done. The general approach of a conversational interface was analyzed and common dialog flow patterns were considered.

Due to that fact, that it had to be a standalone solution, a user interface had been designed and implemented. To build a stable and easy adoptable client a modern Javascript framework was used. The most popular front-end Javascript frameworks were discussed and evaluated and Angular was chosen for the implementation. Furthermore a styleguide was designed to ensure consistency.

To provide natural language understanding several platforms was compared and evaluated. To provide this feature for the Searchchatbot the platform dialogflow was integrated and a parsing strategy was developed. Several intents were analyzed and custom entities were created.

To accomplish a communication layer between the messenger client and the natural language processing platform dialogflow a back-end was implemented. To unify the back-end and front-end programming language node.js was chosen as a back-end solution. Express and Hapi, two node frameworks, were evaluated and in terms of the underlying requirements Express was used.

After receiving of a feedback an evaluation of the user experience was done. The feedback process delivered revealing aspects. Positive and negative survey results of the prototype was discussed and solutions respectively improvements was suggested. From the knowledge of the feedback the most important improvement would be to provide further data source API's to enhance the quality of the results and responds. In general there are positive user signals to continue the work for the Searchchatbot.

Since the importance of chatbots will still rise this prototype is an interesting first step to provide this kind of application for the TU Graz. Natural language platforms are becoming smarter very fast, so also in this area there should be adjustments and further development regarding the Searchchatbot. The bot has also an implemented experimental feature, namely a possibility for voice recognition. Due to the fact that it is not very stable in combination with dialogflow API it is deactivated at the moment but can be activated in the future when the API improves. Alexa and co show the importance of well defined voice interfaces, so it should also be adopted for the Searchchatbot.

This theses showed a possibility of a searching solution in terms of a chatbot and the possibilities how to manage that. All together it can be said that there is an acceptance of it but there are still things to improve since this is an prototype. One can be sure that chatbots itself will play a important role of future application development as Nadella said "Bots will be the new apps"[Cava, 2016].

Bibliography

Banks, Alex and Eve Porcello (2016). *Learning React. Functional Web Development with React and Redux*. first edition. O'Reilly Media (cit. on p. 32).

Bilby (2008). *Turing Test*. Accessed 19.4.2018. URL: https://commons.wikipedia.org/wiki/File:Turing_Test_version_3.png (cit. on p. 15).

Brandtzaeg, Petter Bae and Asbjørn Følstad (2017). "Why people use chatbots." In: *4th International Conference on Internet Science* (cit. on pp. 5, 7).

Brett, John (2016). *Getting Started with hapi.js*. Build well-structured, testable applications and APIs using hapi.js. Packt Publishing Ltd. Livery Place (cit. on pp. xvii, 52, 53, 55).

Cava, Marco della (2016). *Microsoft CEO Nadella: 'Bots are the new apps'*. URL: https://www.usatoday.com/story/tech/news/2016/03/30/microsof-ceo-nadella-bots-new-apps/82431672/ (cit. on pp. 5, 80).

Crangle, Colleen E (1997). "Conversational interfaces to robots." In: *Robotica* (cit. on pp. 16–18).

dialogflow (2018). *Intents*. URL: https://dialogflow.com/docs/intents (cit. on p. 56).

Eschweiler, Sebastian (2017). "Learn Redux - Introduction to State Management with React." In: Accessed 19.4.2018 (cit. on p. 36).

Filipova, Olga (2016). *Learning Vue.js 2*. Learn how to build amazing and complex reactive web applications easily with Vue.js. first edition. Packt Publishing Ltd. (cit. on p. 39).

Freitas, Eduardo and Madan Bhintade (2017). *Building Bots with Node.js*. Automate workflow and internal communication processes and provide customer service without apps using messaging and interactive bots. Published by Packt Publishing Ltd. (cit. on p. 4).

Hezbullah Shah, Tariq Rahim Soomro (2017). “Node.js Challenges in Implementation.” In: *Global Journal of Computer Science and Technology: E Network, Web and Security* (cit. on p. 51).

Horton, Adam and Ryan Vice (2016). *Mastering React*. Master the art of building modern web applications using React. Packt Publishing (cit. on p. 34).

Klopfenstein, Lorenz Cuno et al. (2017). “The Rise of Bots: A Survey of Conversational Interfaces,Patterns, and Paradigms.” In: *the 2017 Conference* (cit. on p. 4).

Kunz, Gion (2016). *Mastering Angular 2 Components*. Learn to build component-based user interfaces of the future using Angular 2. first edition. Packt Publishing Ltd. (cit. on p. 31).

Luka Bradeško, Dunja Mladenić (2012). “A Survey of Chabot Systems through a Loebner Prize Competition.” In: *Conference Paper* (cit. on pp. 4, 14).

Mardan, Azad (2014). *Pro Express.js*. Maser Express.js for your development. first edition. Apress (cit. on p. 52).

Munford, Monty (2016). *How chat apps are transforming the global conversation*. URL: `http://www.bbc.com/news/business-37153831` (cit. on p. 5).

Nimavat, Ketakee and Prof. Tushar Champaneria (2017). “Chatbots: An overview Types, Architecture, Tools and Future Possibilities.” In: *IJSRD - International Journal for Scientific Research and Development* (cit. on pp. 9–12).

Redux (2018). *Motivation*. URL: `https://redux.js.org/introduction/motivation` (cit. on p. 36).

Shevat, Amir (2017). *Designing Bots. Creating Conversational Experiences*. first edition. O’Reilly Media (cit. on pp. 3, 4, 6, 8, 9, 12, 16, 18).

Stapleton, Jennifer (2002). *DSDM: Business Focused Development (Agile Software Development Series)*. Pearson Addison Wesley Prof (cit. on p. 21).

Vepsäläinen, Juho (2016). *SurviveJS - Webpack and React*. From apprentice to master. Leanpub (cit. on p. 44).

Weizenbaum, Joseph (1978). *Die Macht der Computer und die Ohnmacht der Vernunft*. From apprentice to master. Suhrkamp Verlag (cit. on p. 3).