HTW Berlin
University of Applied Sciences

Bachelor's Thesis

Chatbots: Development and Applications

Author Jorin Vogel
First Examiner Prof. Dr. Gefei Zhang
Second Examiner Prof. Dr. Carsten Busch

International Media and Computing
Faculty IV

July 27, 2017

Abstract

Chatbots: Development and Applications explains what chatbots are, what they can be used for and how to develop them.

Chatbots receive increased attention from media and industry, but at the same time it is not yet well known what chatbots really are, what they can be used for and how to create them.
The goal of this work is to answer these three questions by analyzing existing platforms, products and technologies, and additionally developing an exemplary chatbot.
Explaining what chatbots are, demystifying what to use them for and showing how to create them will help more people to be able to use and create chatbots and thereby accelerate the development of the chatbot ecosystem.
Starting by defining fundamental terms, the first half of the work focuses on showing available platforms, products and technologies, while the second half guides through the development of an exemplary chatbot, including user interaction design and software architecture.

Contents

List of Figures

1 Introduction

This work gives a general introduction to chatbots by explaining what they are, what they can be used for and how to develop them.
No previous domain-specific knowledge is required.

Lately as of writing topics around chatbots have received increasing attention from media and also numerous investments from different actors in the industry.
At the same time not many potential users know about the existence of chatbots or about areas in which chatbots could be helpful assistance. The topic is equally unknown to developers.
While the term *chatbot* is commonly used in media, the meaning mostly remains ambiguous. There is a need for further explanation of what chatbots are and further analysis to identify well suited applications for chatbots.
Additionally to spreading knowledge about the potentials of chatbots and their use cases, more developers should be enabled to create new, innovative chatbots.

The lack of knowledge can be solved by providing answers to the questions of what chatbots are, what benefits they bring and how to create them.
An appropriate definition of chatbots can be given by analyzing the fundamental meaning of the term *chatbot* and by exploring past and current applications.
Use cases of chatbots can be identified by exploring existing applications. Additionally, market trends and attributes of media and technologies can be analyzed to find new potential scenarios for the usage of chatbots.
Development is best explained by creating a real chatbot and by using it to present the general principles of the development process.

Explaining what chatbots are, demystifying what to use them for and presenting how to create them, will help more people to be able to use and create chatbots, and thereby, accelerate the development of the chatbot ecosystem.

Innovation in technology and the creation of new solutions can help automating and simplifying more tasks, which gives people the opportunity to focus on more interesting issues and accomplish more things.
Chatbots have the potential to simplify and automate many existing tasks and thereby accelerate the overall technological progress.

The structure of this work follows the three main questions.
To begin with, terminology is defined and applications are explored to form a definition and understanding of what chatbots are.
Afterwards use cases of chatbots are identified not only through the collection of existing examples, but also through the exploration of future potentials by analyzing attributes of the relevant technologies.
The second half of the work is a case study for the development of a chatbot. The presented example guides through the process of designing user interactions for a chatbot, and additionally explains architectural decisions and technological choices, which provide a basis for other developers to build on when creating new chatbots in the future.

2 Fundamentals

Before analyzing a topic it is necessary to have common definitions of its vocabulary. This chapter helps aligning previous assumptions.

2.1 Definition Chatbot

The term *chatbot* consists of two other terms - *chat* and *bot*. The meaning can be better understood by examining the two components separately.

The Oxford Dictionary defines **chat** as "an informal conversation" and more specifically as "the online exchange of messages in real time with one or more simultaneous users of a computer network" [1].
As apparent in this definition, conversations play a central role in *chat* and therefore *chatbots*. Other noteworthy aspects of this definition are the inherent *informal* format of a *chat*, and the traits of being *online* and *real time*.

Informality does not have to be seen as a strict requirement; however a chat message and, for example, a classical letter have different degrees of formality.

Being *online* and thereby not bound to a specific geographic location, device or other physicality can be seen as critical foundation for determining potential types of systems suitable for such media.

The aspect of limiting communication to *real time* implies restrictions on possible interactions and sets a baseline for the expected user experience. This also excludes the usage of certain technologies which do not support the desired responsiveness.

A *conversation* is defined as "a talk, especially an informal one, between two or more people, in which news and ideas are exchanged" [2].

Fundamental to this definition is that there are always at least *two* parties involved in communication and that information is *exchanged*. Keeping that in mind, the kind of systems involved in this should always receive and provide information; chatbots can not work with solely unidirectional interaction.

Bot is defined as being "(chiefly in science fiction) a robot" with the specific characteristics of representing "an autonomous program on a network (especially the Internet) which can interact with systems or users, especially one designed to behave like a player in some video games" [3],

Foremost this provides the information that *bots*, including *chatbots*, are *programs*. The creation of a chatbot implies the creation of an artifact in the form of a computer program.

Furthermore the aspect of *autonomy* and the communication over a *network* can be connected with the previous described trait of a *chat* to be *online*.
The program is given autonomy by not being bound to any specific device. Building on this allows for different solutions than a scenario where the user is in full control of a program's behavior.

Lastly there is a hint in this definition pointing out that a *bot* can often be seen as a *player* in a *game*. This trend towards *game-like* mechanisms and the previous mentioned *informality* suggest the utilization of playful interactions.

Concluding from the combination of these definitions, a chatbot can be defined as an autonomous computer program that interacts with users or systems online and in real time in the form of, often play-like and informal, conversations.

2.2 Conversational Interfaces

Having defined the concept and character of chatbots, one apparent attribute of the technology under examination is that its domain of application involves interaction with one or more users.

Any form of interaction requires an *interface* as a way to interact. An **interface** is generally described as a "point where two systems, subjects, organizations, etc. meet and interact" and in the area of computing it can be further defined as a "device or program enabling a user to communicate with a computer" [4].

This definition leaves a broad array of possible manifestations. However, the range of suitable means of communication can be further narrowed by including another aspect of the previous definition, namely that interaction happens in the form of *conversation*.

Many communication mechanisms can be excluded by focusing on this characteristic of the interaction.
The term *conversation* strongly suggests the usage of natural human language as the base for interaction while discouraging the idea of using an interface consisting purely of static or graphical elements.
Further, conversations are not limited to written language. Verbal conversation is also a possible interface for communication.

Focusing on written, text-based communication, it becomes apparent that not all text-based interfaces fit the characteristics of a *conversational interface*.
Classical command line interfaces, which are often used to interact with computers, are one example of text-based interfaces that do not have the attributes of conversational interfaces. As implied in the naming, these interfaces use *commands* for interaction. A **command** is an "authoritative order" [5] which contrasts with a conversation.
The term *conversation* has a connotation of complex, non-linear communication where each involved party understands the underlying ideas communicated opposed to merely receiving the characters the words consist of.

Understanding of the intentions a user has and the ability to adjust interactions accordingly, sets conversational interfaces, and thus chatbots, apart from other text-based interfaces.

3 Applications

This chapter explores different applications of chatbots to understand the characteristics of a chatbot in practice and to see the kind of applications chatbots can be used for.
Starting with past work and looking at the present in subsequent sections, different approaches and products are presented.
Furthermore, todays ecosystem is portrayed, including an overview about available platforms, existing products and current approaches for the creation of chatbots.
Lastly aspired goals, potentials and promises of chatbots are further identified.

3.1 Past Work

Before exploring new technology one should examine prior work and learn from past ideas, both succeed and also failed attempts.
This section presents a selection of events from the last century, which introduced the ideas that formed the present definition of a chatbot.
It should be noted that this is not an attempt to give an all-encompassing overview about the history of computing, instead the aim is to explain where the concept of chatbots and the interest of creating them originated from.

3.1.1 The Turing Test

Even before the term *chatbot* was coined people started working on machines that interact with humans through natural language.
A first milestone was the 1950 paper *Computing Machinery and Intelligence* by Alan Turing [6].
The ideas he formulated back then are still fundamental to the concept of a *chatbot* in today's world and his thoughts are still central to many discussions about artificial intelligence.
The most famous idea from this paper is the so called *Turing Test*, which is meant to decide whether a machine posses human-like intelligence or not.
Originally Turing called the test *imitation game* whereas the experiment consists of a human interacting with two parties via *textual messages*. One of the parties is another human and

one is a machine. The test subject does not know upfront which party is a machine and which one is a human, but only that one of them will be a machine. During the *game* the human interacts with the other party only through textual messages, but is free to use any variation of messages.
If the human is not able to tell which of the two parties is a machine and which one is a human, the machine passes the *Turing Test*.

When creating a chatbot or another kind of artificial intelligence this test can still be applied to test the *human-likeliness* of the created machinery.
While Alan Turing did not invent the first chatbot, the *Turing Test* was a crucial motivation for the following developments and even today the test still remains to be challenged by new systems.

3.1.2 ELIZA

Fourteen years after the Turing Test was introduced, Joseph Weizenbaum started working on what would be known as the first program to pass a variation of the Turing Test. Joseph Weizenbaum began working at *MIT Artificial Intelligence Laboratory* in 1964 and released the *ELIZA* program in 1966.
Thus *ELIZA* can be considered as the first known creation of a chatbot.

The original version of *ELIZA* was written in a programming language called *MAD-Slip*, which was also created by Joseph Weizenbaum himself, and ran on the *IBM 704* computer. *ELIZA* creates responses to messages that a user inputs via a text-based terminal.

The most famous implementation of *ELIZA* is called *DOCTOR* and simulates a *Rogerian psychotherapist*.
Rogerian psychotherapy is a person-centered therapy intended to let the client realize their own attitudes and behavior [7]. Although relying on mostly simple methods, it remains a popular treatment. Most answers the therapist gives are questions for further details about information which the client mentioned previously. Furthermore clients mostly keep the assumption that a therapist has specific intentions even when asking non-obvious questions.

ELIZA takes advantage of the structure of the English language; the program takes apart sentences via pattern matching and keywords, and reuses phrases after substituting certain

words.
For example, a client's answer "Well, my boyfriend made me come here." can be transformed to "Your boyfriend made you come here?" [8].
Certain signal words and also sentences containing no signals words can be answered with generic, static phrases. Detecting the signal word "alike" in the sentence "Men are all alike." *ELIZA* could pick the programmed phrase "In what way?" as answer [8].

Knowing about the nature of Rogerian psychotherapy, Joseph Weizenbaum created *ELIZA* initially intended as a parody to demonstrate the simple behavior necessary for imitating this therapy.
He was surprised that even people who knew about the inner workings of the program ended up having serious conversations with *ELIZA*. In one anecdote Joseph Weizenbaum told how his secretary, after starting a conversation with *ELIZA,* asked him: "Would you mind leaving the room, please?" [9, p. 5].

Led by the success of the experiment he published the book *Computer power and human reason: from judgment to calculation* in 1976, which presents his thoughts about artificial intelligence, including the differences between machines and humans and the limits of computer intelligence.
In the book he admits that he had not realized "that extremely short exposures to a relatively simple computer program could induce powerful delusional thinking in quite normal people" [10].
This idea coined the term *Eliza Effect* which describes, that people quickly assume computers to behave like humans. This term is still in use today.

3.1.3 After 1970

Another famous program was published by the psychiatrist Kenneth Colby in 1972.
He created *PARRY* as an attempt to simulate a human with paranoid schizophrenia. The implementation of *PARRY* is far more complex than *ELIZA*, but it also models a personality including concepts of how to conduct conversations.
The most famous demonstration of *PARRY* was at the first *International Conference on Computer Communications* (ICCC) in 1972 where *PARRY* and *ELIZA* had a conversation with each other [11].
Later on in scientific experiments *PARRY* also passed a version of the *Turing Test*.

Further programs that have been created to pass the *Turing Test* and which gained the public's attention include
Jabberwacky which was started in 1988 and attempted to learn from the user's input [12],
the in 1991 as an *ELIZA*-like demonstration for a sound card released *Dr. Sbaitso* which was one of the first chatbots for MS-DOS based personal computers [13],
and *A.L.I.C.E.*, which has been released in 1995 and became famous for its realistic behavior, that is based on heuristic patterns instead of static rules [14].

The origin of the term *chatbot* itself can be seen in a paper called "ChatterBots, TinyMuds, and the Turing Test: Entering the Loebner Prize Competition" published by Michael L. Mauldin in 1994, whereby *chatbot* can be seen as a variation of the original term *ChatterBots* [15].

Up until today the *Turing Test* has only been passed limited to certain domains, and there is no chatbot yet that is able to simulate general human behavior indistinguishably from real human beings.
It needs to be noted that, although creating an as human-like as possible system remains a popular challenge, not all applications of chatbots benefit from this type of behavior. Many systems are instead optimized to provide quick and efficient interactions and behave accordingly without attempting to hide their artificiality.

3.2 Present Day

With more than sixty years of history the concept of chatbots is not a recent discovery.
People have been fascinated with the idea of being able to *talk to computers* for a long time; but past attempts have mostly been simple experiments or applications focused on the aspect of entertainment associated with machines inspired by *science fiction*.

Lately the technology industry and press are increasingly interested in the topic of conversational interfaces and chatbots.
The reinforced interest can be explained by observing recent developments of technology and current market trends.

Since Apple's release of *Siri* [16] in 2011 customers have become more aware of the possi-

bilities of conversational interfaces. Even though the capabilities were limited at that time, functionality improved quickly in the following years, which can be attributed to the new competition Apple triggered in the market.

At the same time artificial intelligence gained new traction due to the success of using *artificial neural networks* for machine learning.

The concept of artificial neural networks "dates back to the 1950s, and many of the key algorithmic breakthroughs occurred in the 1980s and 1990s" [17], but only now they are successfully applied.
This is mainly due to the increased computing power available today. A second crucial condition is the necessity for a large disposable amount of data; big Internet companies specialize on collecting data, originally intended to better target advertisement, but now they can use their data to train artificial neural networks.
The technology is named *artificial neural network* because they are modeled after neural networks in the human brain; instead of specifying rules of what a program should do, the machine learns from examples in similar ways to how humans learn. Some tasks that are too complex to be solved with a rule-based program, can now be solved by collecting enough example data, letting the machine figure out the solution instead.
These techniques can also be applied in the field of natural language processing, which is essential to understanding and generating text for conversational interfaces.

With new technical possibilities more people see *conversation as an interface* not only as an idea of science fiction movies, but instead as something that could be possible in the real world.

Also recent market trends make chatbots more compelling.
"Computing is rapidly shifting to mobile devices" [18] and "messaging apps have surpassed social networks in monthly active users" [19]. As a result of these developments users do not have space for complex interfaces on the small screens of their devices, they need solutions light-weight in data consumption, and they can not use complicated keyboard shortcuts. At the same time, users are already spending a majority of their time in instant messaging applications and therefore, they are well accustomed to chatting as means of communication.

Originating from the current state of technology the increasing interest in conversational interfaces leads to new platforms, products and applications for chatbots.

3.3 Platforms

Unless software is distributed with dedicated hardware, software products are designed to be executed by and accessed through other software. The underlying software is the *platform* a product is created for.
Products that target operating systems such as *Microsoft Windows* or Apple's *iOS* require users to install necessary executable files on their local system.
Other software uses the *web* as platform, whereby customers use a *web browser* to access the software over the Internet, while the software itself is executed on another computer referred to as *server*.

In the case of chatbots, the target platform can be any medium that allows users to send messages to each other. A chatbot can be seen as a counterpart to interact with in the same way a user interacts with a human counterpart.

There are numerous platforms available that fulfill these requirements.

While messaging platforms provide means of communication, chatbots function similar to software accessible via a web browser; a server executing the chatbot software is needed and the messaging platform communicates with the server in the same way a web browser communicates with a server on the user's behalf.

Because of the wide variety of available messaging platforms it is not possible to create an all-encompassing collection of available platforms in the context of this work; the following is an overview over the currently most popular platforms, including their capabilities and their area of focus.

One of the most used online communication platforms is *E-mail*.
E-mail however does not have the characteristics of chatbots, defined in 2.1 on page 3, to be able to communicate *informally* and in *real time*; which disqualifies *E-mail* as a platform for chatbots, even though in practice many use cases of chatbots overlap with the ones that can be solved with the automation of *E-mail*.
Although users can choose to express themselves less formally and certain E-mail providers deliver E-mail in a very short time period, this statement is based on the current general use case whereby these two attributes are not given. Still, it is indeed possible for these

characteristics to change in the future and nothing fundamental about the E-mail protocols is preventing their usage for chatbots.

A well-known communication technology, which is suited for chatbots, is *Short Message Service*, or *SMS*.
SMS is primarily used on mobile devices and users are identified by their phone numbers, wherefore the communication has to happen through cellular network providers. However the technology is limited in number of characters, often users are charged by amount of messages sent and communication is limited to text-only. "End of year 2009 user level for SMS globally was 78%, ie 3.6 Billion" [20] users worldwide, which means it remains one of the most popular communication channels; and it therefore is an interesting option for applications with a wide-spread audience or applications requiring a low entry barrier.

Since chatbots can communicate not only via text, but also using voice, *phone calls* are a possible medium too. They are a common way of communication available to a large number of people.
However, for a chatbot that relies solely on voice for communication without any visual feedback, the design of the user experience has to be thought out especially careful. Furthermore to not only understand and generate natural language, but to also parse and generate voice comes with further technical costs.

Apple's *Siri* is another voice-based system available; but as of writing it is not accessible as platform for external services.
Voice-based systems that can be targeted as platforms are Amazon's *Alexa* and *Google Assistant*. Both systems are *general assistants* helping the user with a variety of tasks and in both cases tasks can be delegated to third parties.

Currently popular target platforms for chatbots are *messenger platforms*.
They are primarily text based, they mostly come without cost for end-users and additional to text they often support multimedia formats such as pictures, audio, locations and stickers. Some platforms also allow developers to display sliders, buttons and other graphical interface elements, which can help guiding users instead of exclusively relying on text for communication.
At this point it is not feasible to create a comprehensive list of available features for each platform, since the field is innovating constantly and many of the platforms add new features

almost every single month.

Facebook's *WhatsApp* is with one billion active users in January 2017 one of the most popular messenger applications [21]. It, however, does currently not allow automated access to its platform and therefore using it as a chatbot platform is not a viable option.

The second messenger application belonging to Facebook, called *Facebook Messenger*, is equally popular with one billion active users in January 2017 as well [21]. Contrary to WhatsApp, Facebook Messenger provides a platform for developing chatbots.

Following in popularity [22] are two messenger platforms which are mainly popular in China, *QQ Mobile* and *WeChat*. Both of them currently do not provide a specific chatbot platform, but there have been successful attempts at creating chatbots for these platforms [23].

Further popular messenger applications include the Japan-focused *Line*, Microsoft's *Skype*, *Telegram* and the more business focused *Slack*. All of these applications provide dedicated platforms for the development of chatbots.

The choice of platform primarily depends on the target market. Different audiences prefer different platforms and therefore in certain scenarios one product might be better suited than another.

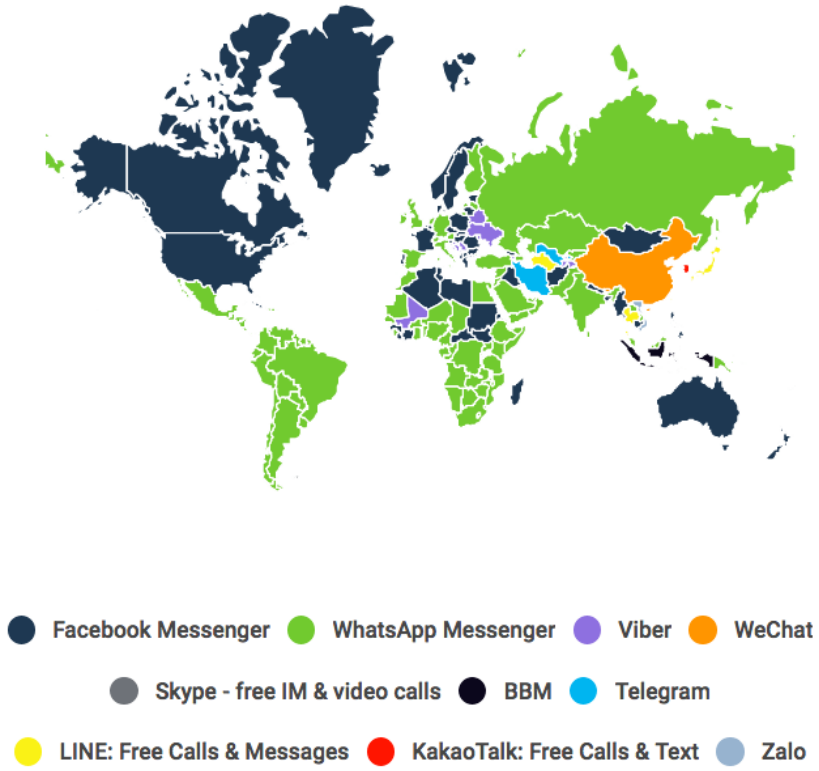

Figure 3.1: Most Popular Messaging App by Country [24]

One important factor can be the geographical location of the target audience.
As visible in figure 3.1, *Facebook Messenger* and *WhatsApp* are the global leading messengers, and as previously mention the markets in China and Japan are dominated by *WeChat* and *Line* respectively, but the data shows some lesser know trends; for example the *Thai* market is also dominated by *Line* and in *Iran*, *Telegram* is the most popular messenger application.

3.3.1 Cross-platform Development

As with the creation of other kinds of software, it is possible to release the same chatbot software for multiple target platforms, whereby the interaction between software and platform has to conform to the technical details and protocols of each environment, the usage of platform-specific features has to be adapted individually and the user experience needs to be designed to fit each environment's expectations.

There are existing frameworks that allow developers to develop a chatbot once and release it

to multiple platforms at the same time without any adjustments to individual platforms.
One such framework is *API.ai* by *Google*. As of writing it supports 16 different integrations, including platforms such as *Facebook Messenger*, *Skype* and *Slack* [25]. However, this platform is more than an adapter to different platforms. It is a complete solution to developing chatbots. *API.ai* comes with built-in support for natural language processing, a chatbot is already able to have basic conversations out of the box, and developers can train chatbots about new topics by just providing example conversations while the framework handles all of the language parsing.
Detected keywords and *intends* can be forwarded to be handled with custom logic; although for simple use cases this might not be necessary and a chatbot can be developed without writing a single line of code.

However, there are also limits to platforms like *API.ai*.
First, *intend parsing* is, in the case if *API.ai*, currently limited to a finite list of topics. If a chatbot is handling topics from a domain unknown to *API.ai*, this solution is not sufficient anymore.
Further, developers have no control over the applied machine learning and natural language processing algorithms. There are no possibilities for customization, if the parsing results or the generated responses do not match the requirements.
Additionally, while *API.ai* currently supports 15 different languages, a chatbot is limited to these available languages. If another language needs to be supported, a lot of work might be necessary because the system needs to be recreated with a custom solution.
Moreover, an issue to keep in mind is, that, while *API.ai* has support for many platform-specific features such as custom formats for message content and *quick reply* buttons, there are unique features of single platforms that are not supported and since the space is evolving at such rapid pace future extensions might not be available either.
Even though *API.ai* is at the moment free to use for everyone, they will very likely search for a sustainable business model in the near future. The business could be sustainable by charging for the service, collecting data which Google can use elsewhere, binding customers to their services and gaining market share.
Regardless which monetization strategy is chosen, as a developer one has to be aware that such a service is a non-controllable, external dependency.

3.4 Communication Mechanisms

Nowadays there are two fundamental ways a chatbot can communicate with users; some platforms provide user interface elements, such as buttons, that can be displayed to users; otherwise communication is done solely with natural language.

Interface elements limit user input to a number of predefined actions, while natural language has no restrictions on possible inputs. By suggesting possible actions in the form of a list of buttons or similar, possible interactions become obvious and simpler for the user. Even when limiting interaction to certain actions, the main characteristic of a chatbot remains; the interaction is still structured in the form of a conversation, only that the choice of input is restricted.

In certain scenarios there are too many possible user inputs to fit in a fixed list. In these cases natural language is a more appropriate input method.
Custom input requires to be parsed to extract information. As an example, when the user is asked for a date and time, and repeating times are also allowed, a user could express a time as *"every second Tuesday at 6am and 9pm"*, while in a traditional user interface this would require a non-trivial amount of interface elements.

If natural language is used for communication, it should be clearly stated what kind of input is expected, so that the user knows which topics and which variations of input the system understands.

One of the main challenges with a natural language interface is handling the non-restricted interaction in a coherent way; since user input is in no way limited to a single topic, all sorts of unexpected user reactions have to be accounted for. As a consequence of this, there are certain conversations every chatbot has to be able to handle in some way. This includes, but is not limited to, simple smalltalk such as questions like *"How are you today?"*.

Most platforms allow for a combination of the discussed communication mechanisms; predefined actions can be displayed as suggestions to the user, while the user is also able to use natural language.
By combining both methods it can be taken advantage of the benefits of both; there is a clear guideline for interactions and, simultaneously, the user is free to express any possible custom input.

3.5 Classification of Chatbots

With the increasing number of messaging platforms opening up for chatbot development, companies have become interested in releasing their product for this new format and some companies also create new products focusing solely on the chatbot market.

It is still a new, not fully formed market, but there are certain trends for what companies are interested in creating.

One helpful classification of chatbots is categorizing them in terms of features they provide. The following categories are adapted from the article "7 Types of Bots" by *Dotan Elharrar, a Product Manager at Microsoft AI & Research* [26].

Single-feature Chatbots A large number of chatbots provide only one single feature. These chatbots are limited in functionality but simple to use. One example is a Facebook chatbot called *Instant Translator* [27]; in the beginning the user selects one language to translate to. From there on *Instant Translator* simply translates all text it receives into the selected target language.

Proactive Chatbots This category describes chatbots which push information to the user instead of answering questions in conversations. Hereby the user does not need to interact with the chatbot, but only uses it as service to receive information at certain times. One example would a service which sends the user a daily weather forecast. Another use case is the chatbot for Facebook Messenger from the airline *KLM* [28]; users can use the service to get updates and information about their booked flights.

Group Chatbots There is a range of functionality chatbots can provide when they interact with a whole group of people instead of only a single user. These chatbots are limited to platforms which provide the necessary features to use chatbots in group conversations. A simple example for a *group chatbot* is called *Roll* [29] for a messaging platform called *Kik*; when sending a question to *Roll*, the chatbot answers with a random name picked from the members of the group.

Simplification Chatbots In a few cases chatbots are used to provide users with a simpler interface to complicated existing tasks, which would traditionally involve many bureaucratic

and formal steps.
One example is a service called *DoNotPay*. It is advertised as "the world's first robot lawyer" [30] and the service helps the user with simple legal problems, such as fighting parking tickets.

Entertainment Chatbots A popular kind of chatbots are still chatbots whose functionality consists only of having conversations with users. These services don't interact with other resources apart from the conversation itself. The in 3.1.2 on page 7 described *ELIZA* belongs to this category.

Personal Assistants This category consists of chatbots that combine many different features and can be seen as platforms of their own. *Siri* and *Alexa*, which have been mentioned earlier, belong to this category.

Optimization Chatbots This category tries to make existing products more accessible by creating a chatbot for users to connect to a product.
The difference to a *simplification chatbot* is that a *optimization chatbot* is not built on an external entity, such as the legal system of a state, but it instead connects to a product a company has full control over.
By taking advantage of new platforms, companies like to reduce friction for customers to use their products. The currently most obvious aspect of chatbot platforms is the ease for users to access products. Companies like to optimize the use of their products by making them available via the conversational interfaces of chatbots.

Use cases fitting the category of *optimization chatbots* can be found across many different industries. The article "100 Best Bots For Brands & Businesses" [31] lists examples from different industries using chatbots to optimize access to their products.
Products include beauty brands such as *Sephora*, consumer goods like *Johnnie Walker*, entertainment companies including *Disney* and *Marvel*, fashion brand such as *H&M*, financial services like *PayPal*, food delivery from stores such as *Pizza Hut*, E-commerce platforms including *eBay*, traveling services such as *Airbnb* and *Expedia*, airlines like *Lufthansa* and *British Airways* and many news outlets including *Washington Post*, *New York Times*, *Forbes* and *BCC*.

As apparent from the engagement of many well established companies, brands are very

interested in being present on messenger platforms.

While there is growing interest in targeting messaging platforms as new markets, currently user engagement with available chatbots has not reached the popularity of other channels such as mobile applications yet, and most chatbots created so far do not provide much functionality, but instead redirect users to their existing products.

With increasing interest, engagement and financial investments, the arise of more sophisticated products can be expected in the future.

3.6 Promises

As explained previously in 3.2 on page 9, the interest in chatbots and conversational interfaces increased in recent time mainly driven by the popularity of messaging platforms, mobile devices, and personal assistants, and also by the advancements made in the field of artificial intelligence.
The conditions are right to think about taking advantage of the possibilities in the newly available market, however, the question remaining is what can chatbots achieve that existing solutions are not good at, both, from a user's point of view and looking at the interests of companies.

From a user's viewpoint chatbots can be seen as new interface to interact with computers. Existing interfaces are often not intuitive for humans. Humans need to initially learn how to use the technology. With every new application one installs and every new website one visits, there is a new interface to adapt to.
"Adjusting to a machine does not come naturally to us. With every app you need to learn how to use it. ... Conversations come naturally to us" [32].
Conversation is a way of communicating that humans already know how to use. It is the fundamental method for humans to interact with other humans. If it is possible to use this communication technology also to interact with machines, the interface would be by default intuitive for humans to use. "The vision for a chatbot: get machines to respond to questions like a human being" [32].

Further, there is a trend in consumer behavior of "outsourcing their "chores", such as driving, shopping, cleaning, food delivery, errands" [33] to companies that offer these services.

Service companies are not a new occurrence, however in the past it took more effort to coordinate the usage of such services. By using technology to automate many steps of the coordination process, not only the cost can be lowered, but also the friction for customers to use a service is reduced significantly. Managing and coordinating the usage of such services are tasks conversational interfaces are particularly suited for because this is a scenario that profits especially from the simplicity and low friction that characterize conversational interfaces.

Users can also profit from technical advantages chatbots have over native applications and websites.
Native applications need to be downloaded upfront, including all their resources and not just the ones required at this moment. Websites are slimmer and only the resources required to load the current page need to be downloaded, but one page still contains not only content, but also layout information, styling, decorative images and in most cases also JavaScript to run some additional logic in the web browser.
The only thing a chatbots needs to download over the network is content. Everything else is provided by the platform it is embedded in.
Compared to existing solutions, "a chatbot uses very low bandwidth" [32], which can be an important advantage not only for the perceived responsiveness, but especially in scenarios with slow network connections.

Providing customers a more intuitive and more direct way of interacting with a company's product is already a compelling reason for a company to be interested in the new platforms. But there are additional benefits companies can draw from conversational interfaces, which are not perceivable by users.

First, "the cost of developing a chatbot is one-third of what is required in developing a mobile app" [32]. This might not be the case for every product but in general, creating a chatbot is less work than creating a mobile application, because neither is custom design required nor is it necessary to write code for the logic controlling the user interface.

Next, "Chat apps also have higher retention and usage rates than most mobile apps" [34]. Since chatbots are part of a chat application they can take advantage of being where the attention of mobile phone users already is. A chatbot can therefore potentially gain more user engagement than a competing website or mobile application.

Another aspect of using natural language as an interface is that "chatbots are able to gain invaluable data and insights on user behavior" [35], because firstly, users have the freedom to send any kind of information and feedback, and secondly, being in the context of an informal conversation people tend to be more talkative than they would be in a more formal environment.
Especially for companies such as media outlets or retailers being able to further profile users can be a useful assistance in tailoring personal experiences for users and targeting them with individual offers.
Additional context and user data is also available on the platform itself; when interacting with a user via a chatbot on the *Facebook Messenger* platform, all public information of the user's *Facebook* profile is also available for the company to use for further personalization.

Lastly, the most fundamental reason for a company to be interested in chatbots is the before mentioned popularity and ubiquity of messenger applications. "The question brands and publishers now face is how to engage with these private social network users" [35]. When the attention of users is shifting away from not only non-digital media but also away from other mobile applications and traditional social networks, companies need to find a way to reach users at the place they spend most of their time at.

3.7 Limitations

While chatbots have many promising properties, chatbots also come with restrictions which frame them as unfeasible solutions in certain scenarios.

One fundamental property of the architecture of chatbot systems, is that they are not software running on the device of the user.
They mimic the nature of chat, that two parties communicating with each other are using two separate devices.
This fundamental design of chatbot implementations is similar to the basic idea of browsing the *world wide web*, whereby a network connection is a mandatory requirement.
Typically network connections are part of the Internet and such a system can be described as purely *online*.

This underlying design has certain implications for the usage of chatbots.

First and foremost, this implies that all scenarios, which are obliged to work without a network connection, cannot make use of chatbots.
Exemplary areas of applications affected by this implication are applications that forbid network connections due to high security standards and also products targeting rural and remote locations with unstable network connections.

Another consequence is that a certain amount of latency is unavoidable with networked applications.
When information needs to be transfered from one device to another, it needs to be transported through more physical space.
For a chatbot based on *Facebook Messenger*, this means information needs to be transfered from the user's device to a data center, where Facebook is operating their *Messenger platform*, further to another server where the developed chatbot software is running, and back to the user through the intermediary data center.
Many factors influence the accumulated latency, of which certain factors can hardly be influenced by the developers and operators of the chatbot software.
Aspects of networking such as the capabilities of the user's devices, the conditions the *Internet service provider* of the user is operating under, *domain name lookups*, *IP package routing* and associated *package loss*, the locations of Facebook's data centers, and the performance of event processing and forwarding of the *Messenger platform*, are only a selection of unknowns when operating a chatbot.
The unavoidable latency and the unpredictable parties involved in the networking layout make it clear, that chatbots are not suitable for any time-critical applications. It can not be guaranteed that a user receives a response within an unnoticeable period of milliseconds or with a delay of tens of seconds.

Not only the dependency on the network is limiting the capabilities of chatbots, but also the very idea of using *chat* as a medium.
While text is undoubtedly a powerful medium, there are certain applications where other media are better suited.
Many messaging platforms already enhance the communication by adding interface elements such as buttons and by allowing multimedia content such as images and videos to be sent. However the linear layout fundamental to text communication remains and its immutable nature limit interactivity.
While chat can be a very simple and efficient way of communicating and it is appropriate

in many scenarios, other use cases are address better with existing solutions such as native or browser-based applications. Obvious candidates, which are better served with other technology are photo and video editing applications and also 3D gaming.

Due to the limitations the medium chatbot has, it can be anticipated that, while they address many needs of human-computer interaction, they are not universally applicable and they are not able to replace all other media.

Much in the same way that the invention of radio has not replaced newspapers and neither did the introduction of television replace radio, chatbots can be seen as a new possible communication concept, but they can coexist with previous technology in a way that each of them focuses on the area they are best suited for.
Understanding not only the merits but also the limitations of a medium is essential for deciding whether the medium is a good fit for a use case or if another medium is a better suited solution.

4 Development

To not only understand where chatbots can be applied but to also understand how a chatbot can be created in practice, this chapter guides through the development of an example chatbot.
The description of the development process is kept on a more general level and many aspects of the presented architecture and solution can be reused when creating other kinds of chatbots.

First of all a suited application needs to be chosen and specified in its requirements. Before starting with the implementation, possible usage scenarios need to be defined and matching user stories have to be created.
When all requirements are set, appropriate platforms, tooling and solutions can be selected. After all preparations are done the technical implementation can take place.

4.1 Choosing a Practical Example

A suitable application for a chatbot needs to be selected before thinking about implementation details.
As illustrated earlier, chatbots can be used to cater a wide variety of applications.

Since this application should be a demonstration of the different aspects of developing chatbots, it should not be too simplistic in scope. An appropriate example covers more than one of the product categories described in section 3.5 on page 17, while being, at the same time, not too technical challenging in a specific problem domain outside of chatbot development.
A service, that accepts image files and returns the same picture with a filter applied, might be an interesting and entertaining use case for a chatbot, however, it would not be an adequate example to give a general introduction to chatbot development since, even though it would be a technical interesting task to solve regarding image processing, it would not illustrate much technical details in the domain of chatbot development.

The here selected example application is a system for **individual language studying**.
While this idea arises from a personal need for such a system and an observed shortage in the functionalities existing solutions provide, this application also fulfills the stated requirements of a suitable example application.

The language studying chatbot covers multiple of the categories defined in section 3.5.
It is mainly an *optimization chatbot* improving the task of language studying, but the chatbot can also be classified as a *proactive chatbot* because it can use proactive features to notify the user when it is time for studying.

Although there is a necessity to understand parts of the problem domain of language studying, the required domain-specific knowledge is minimal.

The idea is to build a system independent from existing learning resources that users can use to study their own vocabulary. Since there is no need to focus on content for specific languages, the main focus remains implementing the chatbot features, which are primarily, that a user is able to input new vocabulary, and then the system tests the user's knowledge in appropriate studying intervals.

4.2 Existing Solutions

When starting a new project, it is helpful to research existing solutions which solve similar problems.
There are already a variety of existing software applications for language studying, which have various use cases and solve different problems.

One significant separation is between software that includes content and software users can utilize and customize to study personal content.

The first segment is the most prominent. This software is intended to enable people to self-study language and, at least partially, replace physical language courses.
A main reason for the prominence of this segment is the ability to sell content. Professionally curating the curriculum for a language course requires teaching expertise and takes a lot of effort. Because of this, good content for language studying remains expensive, not only in software but also in the form of physical textbooks.

One popular example from this segment is *Duolingo*. "Duolingo has courses in a handful of languages.... The courses are structured in a way like games as well - you earn skill points as you complete lessons" [36].
Interestingly, *Duolingo* recently released a chatbot [37] as part of their *iPhone* application which enables the user to have a text-based conversation about certain topics with a chatbot to learn the appropriate phrases for the given scenario. Although the topics and possible phrases are restricted in each scenario, this is a first example of how chatbots can be used for language studying.

The second segment consists of software which does not provide users a guideline for what to study, but is instead intended to support users studying their own vocabulary.
Most of software belonging to this segment are attempts at bringing traditional flashcards to digital media.

One of the most established applications in the second segment is *Anki* [36], which exists for more than ten years already and provides a flexible, but also rather complex, interface to create various kinds of studying material.
Another more recent competitor is *Memrise*, which gives users a more intuitive interface, that also includes several gamification[1] features to make the studying process more appealing.

Both mentioned products are not restricted to a field of study and users are able to add their own content. Furthermore both products also offer mechanisms for users to share content with others, which allows users to reuse what other users created.

For the in this work planned chatbot example the second segment is more fitting, since there are no resources in the current context to curate professional content.

The following implementation is an attempt at creating a software product with a conversational text interface that is in its use cases similar to products like *Anki* and *Duolingo* while making use of the unique features the medium *chatbot* provides.

1 The Oxford Dictionaries describe gamification as "the application of typical elements of game playing (e.g. point scoring, competition with others, rules of play) to other areas of activity" [38]

4.3 Use Cases and Requirements

Building on the analysis of existing solutions in 4.2 on page 25, features for the chatbot need to be specified.
An effective method for gathering crucial features is to find potential users and create usage scenarios for their individual needs.

To apply this method, the fundamental problem the application is solving needs to be defined first.

The issue this chatbot is trying to help with is the study of individual vocabulary. The goal is not to provide studying material in a way a language course or a textbook does, but instead to complement these resources with a tool to study new vocabulary and phrases students pick up while studying or in different situations in every day life.

4.3.1 User Stories

The following are two hypothetic scenarios of individuals that might use the chatbot and both profit from it in different ways.

Clara is a 22 years old American. She moved to New York City to go to university. Currently she is in the last year of her bachelor degree in economics. In university she signed up for an evening class in Mandarin. She uses *Facebook Messenger* every day to talk to her friends and she discovered the chatbot when a friend sent her a link. For her the most difficult part of the studies is to write hànzì, the Chinese characters. Now Clara uses the chatbot to write down vocabulary in hànzì during her class, and at home she revises the new characters by going through them using the chatbot and writing the characters down on paper.

Pierre is 29 years and born in Bordeaux, France. He studied computer science and a year ago he moved to Berlin where he found a job in a startup. At work all communication is done in English since the team consists of people from all around the globe. Because Pierre is not a native English speaker, he picks up new words at work almost every single day. Since moving to Berlin Pierre also made a few German friends and he tries to pick up new words they teach him. He found the chatbot on a news website for technology products, and since

then whenever Pierre learns a new word he grabs his phone from his pocket and adds the word to the chatbot. Since the chatbot has no restrictions on what to learn, Pierre uses it to save both, German and English, vocabulary in one place. Pierre's daily commute from and to work takes 40 minutes twice a day. Now he uses his commute time to take out his phone and review new vocabulary he picked up the previews days.

4.3.2 Functional Requirements

All necessary functional requirements can be extracted from the above defined user stories.

First, a user needs to be able to add new vocabulary.
There should not be any restrictions on what can be added and vocabulary should not be limited to single words because in many cases it is more helpful to add whole phrases instead. Each vocabulary consists of the phrase the user is trying to remember and an explanation to help understanding the meaning of the phrase.

Next, the chatbot requires a way to revise vocabulary.
There should be two possible modes for revising; one version where users can decide on their own if they remembered the phrase correctly and when to go to the next phrase, and a second way whereby users type out the phrase in the messenger application.
In each case the system should keep track of whether users knew the correct solution or not.

Lastly, it is necessary to have a means of deciding what to study next.
A user should not be required to think about what or when to review vocabulary. The chatbot needs a system to decide the review time for each vocabulary, and ideally the user is notified when vocabulary is ready to be reviewed by sending a message to the user.

These three main features can be seen as a sufficient *minimal viable product*, or *MVP*.

For demonstration purposes it is desired to keep the product as simple as possible.
The knowledge that can be taken from making decisions about the implementation and walking through the process of creating the chatbot, is mostly independent from this particular product and can be applied to the development of other chatbot products.

4.3.3 Non-functional Requirements

Since this is a simple example, non-functional requirements remain minimal.

Availability of the service is not a priority, but chatbot software can be scaled similar to other software and redundancy can be used to ensure availability.
Since messaging platforms are an intermediary between users and the chatbot software, most platforms also re-send missed messages in case the chatbot is unavailable, which further lessens the priority of availability.

Similarly, *security* is not a main focus here since the dependence on a messaging platform ensures certain factors such as authentication and encryption of communication.

Performance is equally not a major concern at this point.
Because the scope of the example application is limited and the functionality simple, the domain specific logic is inexpensive in computation.
The main performance bottleneck is the in 3.7 on page 21 mentioned aspect of networking and involved unknown parties.
Employing performance-improving solutions for networking issues won't be a part of the example chatbot, but performance can be improved by choosing geographically strategic located data centers for deploying the chatbot software.

A more central requirement is *reusability*. Although the example is a chatbot solving a specific task, the software architecture should be designed in a way, that appropriate parts can be reused for other chatbots in the future.
To ensure reusability the software should be *documented*, *stable* and *extensible*.

Usability can be seen as the most important non-functional requirement.
The focus of developing the example chatbot is to design a good user experience and to explore how interface and interaction design is work for this medium.

4.4 Technical Setup

After knowing the features the chatbot should support, technical requirements can be extracted and appropriate technology chosen for the implementation.

4.4.1 Communication

As discussed in 3.4 on page 16, there are two fundamental ways for communicating with a user, interface elements and natural language.

Since the previous defined features for the minimal viable product of the example chatbot require not many options, everything can be represented with unambitious interface elements.

The concrete implementation of available actions is shown in the next section. For now it should be sufficient to know that all of them can be represented as predefined actions.
However guiding the user with interface elements instead of natural language does not imply that natural language can not be used complimentary.
Quite the contrary, the example chatbot is only possible by analyzing user input; when adding new vocabulary, phrases and their explanations need to be captured, and likewise when studying the user's guesses need to be evaluated.

The implementation of this chatbot demonstrates the complementing use of interface elements and natural language side by side.

By relying only on simple input parsing and no advanced natural language processing techniques, a major source of complexity of chatbot development can be avoided.
While there are useful techniques that enable previous impossible use cases, they are not necessary to explore the fundamentals and paradigms of chatbot development.

4.4.2 Platform Selection

An important question to answer when developing a chatbot is which platform to target. As shown in 3.3 on page 11 there are many possible target platforms and some of them are fundamentally different to others.

Deciding for a voice-based platform like *Alexa*, for *SMS* or for a messenger platform has consequences for all further decisions.
For the example case, a voice-based interface is less appropriate since users should be able to control the precise spelling of the vocabulary and further, there is currently no voice-based

platform available that supports multiple languages simultaneously.

SMS communication is better suited for scenarios that only need to send a low number of messages, since there are costs for sending text messages.
As of writing, *Twilio*, a cloud communications service, charges $0,0075 for receiving and $0,085 for sending a message when using their *Global Short Message Service API* in Germany [39]. Supposed the chatbot has 1000 active users that all study 100 phrases daily, the costs would accumulate to $277.500 of monthly expenses[2]. These prices are affordable if a company is selling airplane tickets or something similar, but in other scenarios another messaging platform is better suited.

By choosing a messaging application as target platform, there are no monthly costs to be taken care of. Additionally there are further interface elements than only plain text available for interaction. But there are many major existing messaging applications and deciding for one can be difficult.
As shown in 3.3 on page 14, different platforms have geographically different target markets. If a chatbot targets the Chinese market *WeChat* would be the obvious platform to choose; likewise Japan would be targeted by using *Line*.
In North America and Europe *Facebook Messenger* and *Facebook's WhatsApp* are currently the leading platforms. Since as of writing *WhatsApp* does not provide publicly available access to the platform, *Facebook Messenger* is the biggest platform one can currently target. By creating the first version of the example chatbot with English as an interface language, the target markets are primarily North America, Europe and Australia and therefore *Facebook Messenger* is a natural fit as target platform.

As mentioned in 3.3.1 on page 14, there are also solutions to create chatbots using a framework that allows to release a chatbot to multiple platforms at the same time, but as previously noted such a framework also has drawbacks.
With the limitations in mind and with the goal of keeping the example as simple as possible, it will be implemented for a single platform without abstracting the process by using third-party frameworks.

Facebook Messenger is a fitting platform not only due to popularity, but also because it offers

2 $1000 \times 100 \times 30 \times (0,085 + 0,0075) = 277.500$

mechanisms for interacting with chatbots via interface elements, which will be used in the next chapter to display buttons in the user interface.

The registration and configuration of a chatbot for *Facebook Messenger* is not explained in detail here, since Facebook's online documentation already covers detailed explanations and the majority of the settings are specific to each chatbot.
It should be noted that Facebook refers to a chatbot in *Messenger* simply as *bot*, and to create a *Messenger bot* it is required to first create a *Facebook Page* and an *application* for the page. Afterwards *Messenger* can be added as a *product* to the application.
Further information can be found in the developer documentation.[3]

Integration

The development of a chatbot for *Facebook Messenger* is similar to most other platforms.
When creating and configuring a chatbot, the developer registers a *WebHook*. "The concept of a WebHook is simple. A WebHook is an HTTP callback: an HTTP POST that occurs when something happens; a simple event-notification via HTTP POST" [40].
After the setup is done, Facebook will now send *HTTP POST* requests to the registered URL containing event information for every message a user sends to the chatbot.
This way the developer has complete control over handling each message on an arbitrary machine that is reachable via a public URL.

4.4.3 Server Technology

At this point that the interaction with the platform is decided, the application can be created on a custom server, and it has to be decided how to structure the application running on this server.

Programming Language

Since all interaction with the platform happens via HTTP, there is no restriction on which programming language to use as long as it can be accessed through HTTP.
Chatbots can be written in any programming language.
Depending on the specific application, different programming languages are better suited

3 https://developers.facebook.com/products/messenger

than others though.

In the example case there are not many specific technological requirements.
However, to be able to send notifications to users when their studies are ready, the system requires a way to schedule timers. The timers need to be lightweight enough to be re-scheduled for every user activity that can affect the time the notification is scheduled for.
Single-threaded programming languages, such as Python, Ruby or PHP, mostly use separate worker processes to handle scheduled jobs, but in this case there would need to be a worker process for each user waiting until it is time to send notifications to the specific user.
Another way to handle this in single-threaded programming languages is by using a system for *asynchronous, evented I/O*, such as the *asyncio* module for *Python* [41] or the *Node.js* JavaScript runtime [42].

The example chatbot is created with the *Go* programming language [43].
Go is a multi-threaded language which allows for taking advantage of all available CPUs on a machine without the overhead of creating new processes.
It is well suited for scheduling notifications and Go includes a robust web server in the *standard library* which can be exposed to the public Internet without a proxy server, which is a common approach with the previous mentioned programming languages.
With this setup there are less moving parts to take care of and the example can be a simple, self-contained application.

Data Storage

In the example case data needs to be stored and it should be stored locally on the same machine to keep the application simple.

Since a user can save new vocabulary, there needs to be a way to store information for each user.
For the studying itself, further information has to be stored. It is necessary to keep track of correct and incorrect guesses, it needs to be decided when to study next and the time of the last study needs to be tracked too.

To send notifications additional information has to be stored.

It is necessary to know when the user was last active and if the user saw the last notification, since notifications should only be send when the user is not already active at this moment and when the user has already seen the last notification.

All of the required information is always bound to a single user. Users can never share information with each other. There are no further relations in the data.
Without relational data the features that relational databases such as *MySQL* or *PostgreSQL* provide are not used; instead a simple key-value store can be sufficient for storing the data. The example uses an embedded key-value store called *BoltDB* [44], which does not need a separate process and saves data on disk in a single file.

These technical decisions are the base for the following description of the chatbot implementation.

4.5 Features and Interaction Design

At this point it is defined what the example chatbot should be able to do, and which technologies are used for its implementation.

Before looking at the technical implementation, the usage of the chatbot is shown from a user's point of view.
The following is a presentation of the features of the chatbot and the design of the interactions to use these features.

A *Facebook Page* and a *Facebook Messenger bot* have been created under the name *Studybot*.
By not publishing the *Facebook Page*, the chatbot remains only accessible for administrators of the *Facebook Page*.

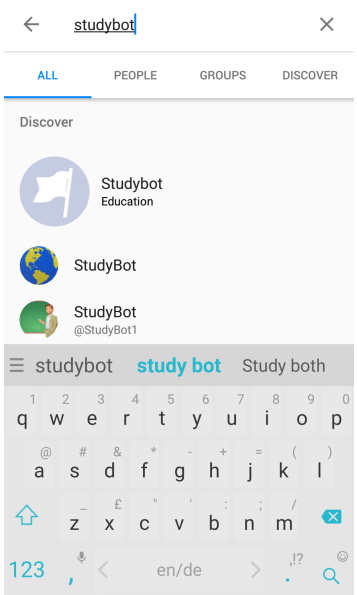

Figure 4.1: Search

The demonstration uses the Messenger *Android* application, but *Messenger bots* can also be accesses through applications on other platforms or using the web version of Facebook.

When using the Messenger application as administrator, the chatbot can be found by using the search as shown in figure 4.1.

After navigating to the chatbot, a description becomes visible. This can be seen in figure 4.2.
It contains the profile image of the *Facebook Page* the chatbot belongs to, the category the *Facebook Page* is part of, and a text describing the functionality of the chatbot, whereby all of these elements can be defined by the developer of the chatbot.
A button labeled **Get Started** is displayed at the bottom of the screen.

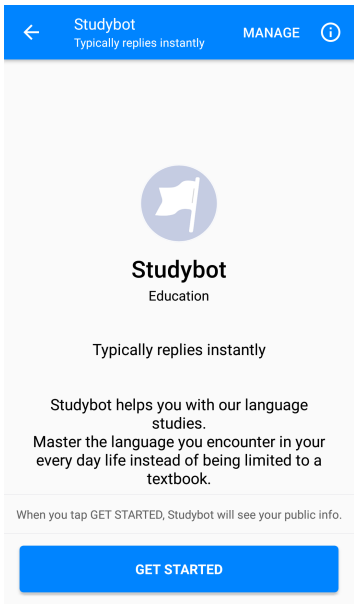

Figure 4.2: Get Started Screen

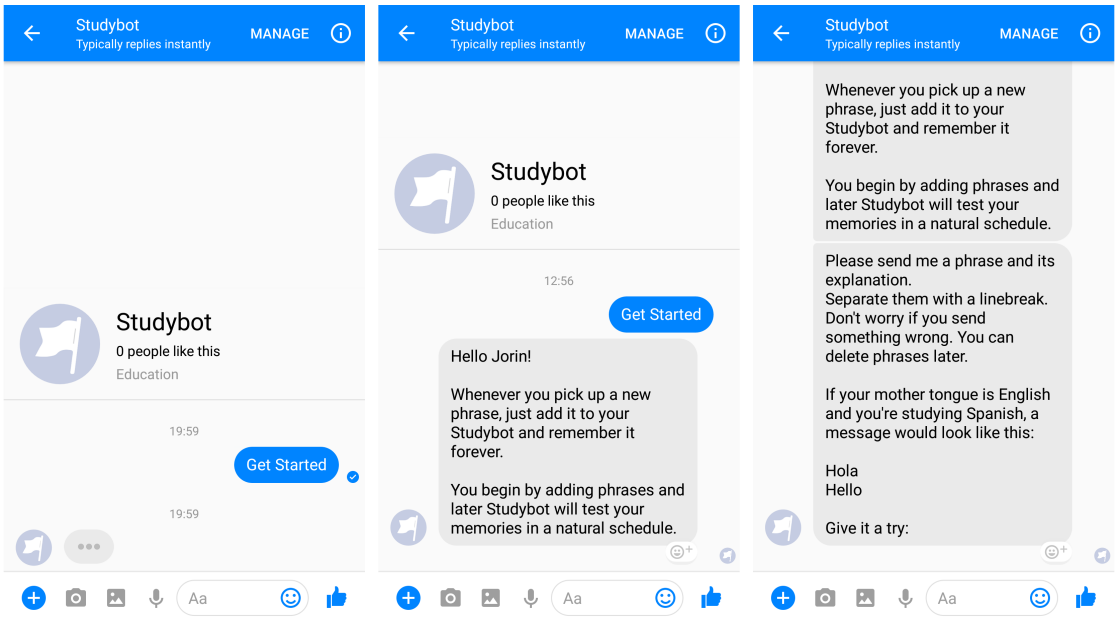

Figure 4.3: Introduction to Studybot

When pressing the **Get Started** button, a message is sent to the chatbot, and as indicated by the *dots* visible in figure 4.3 on the left image, the chatbot is active and about to send a reply.

In normal conversations the *dots* are used to indicated when a user is typing.
For chatbots the *dots* indicate that the chatbot received the message and is crafting a reply.

The image in the middle of figure 4.3 shows the first message sent to users. They are greeted with their first name to create a more personal feeling atmosphere. The greeting is followed by two sentences explaining the chatbot.
When working with text as a medium, it is especially critical to focus on the most important information. In graphical interfaces users can get an impression with a single glance, but to perceive the meaning of text users need to read word by word.
In most scenarios the time users are willing to invest in understanding a product is limited, every word in a text has to be selected carefully.

It can be seen in the image on the right of figure 4.3 that a second message is sent to the user. The second message is delivered 5 seconds later than the first one to not overwhelm the users with too much information at once.
This message contains instructions to begin using the application.
The instructions are additionally illustrated by providing an example for a message in the expected format.

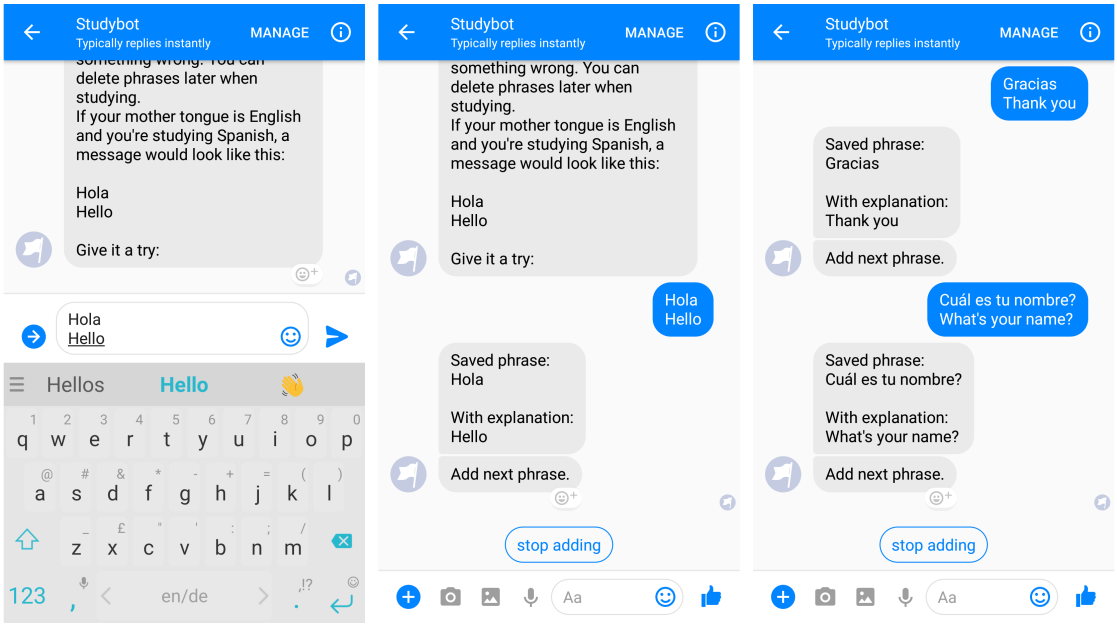

Figure 4.4: Adding three phrases

At this point the user needs to interact with the chatbot.
Figure 4.4 shows how a user adds phrases to *Studybot*. When typing a phrase, users first type the phrase in the foreign language they like to study, then a *line-break* needs to be added, and the next line contains an explanation for the preceding phrase.
After the phrase is sent to the chatbot, a confirmation message is displayed to keep the user updated about the current state of the system.
This is followed by another message prompting the user to add more words.

For a chatbot it is critical to always keep the user informed about the next expected action, therefore most messages should end with a prompt for the next input.

The image in the middle of figure 4.4 has a button displayed at the bottom of the messages, which the user can use as an alternative way for interacting with the chatbot.

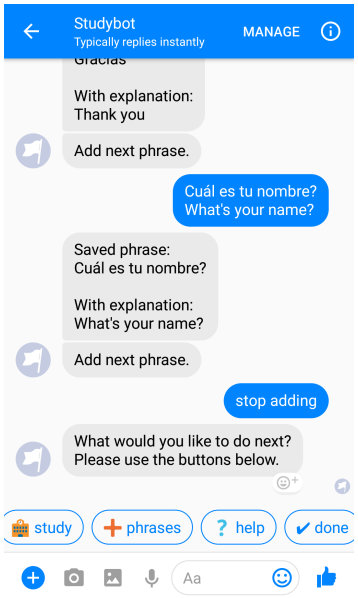

Figure 4.5: Stop adding phrases

As shown in figure 4.4, three phrases have been added to *Studybot* before the user decides to press the button labeled as **stop adding**.
Figure 4.5 shows, that after stopping the adding of phrases, the user is prompted with an array of four possible actions. The actions are labeled as **study**, + **phrases**, **help** and **done**.
Each button label also contains an emoji[4] used as icon for the action to make it easier identifiable for users.
When the usage of graphical elements is limited and one has to rely mainly on text for communication, emojis can be a helpful substitution for traditional icons to provide visual guidance.

The chatbot has different *modes* of interaction.
Internally the chatbot needs to keep track of the current *mode* of each user to be able to

4 "Emoji are pictographs (pictorial symbols) that are typically presented in a colorful form and used inline in text. They represent things such as faces, weather, vehicles and buildings, food and drink, animals and plants, or icons that represent emotions, feelings, or activities." [45]

address each message in the correct context.
In figure 4.5 the user has left the *mode* for adding words and entered the *menu mode* which provides access to other modes.

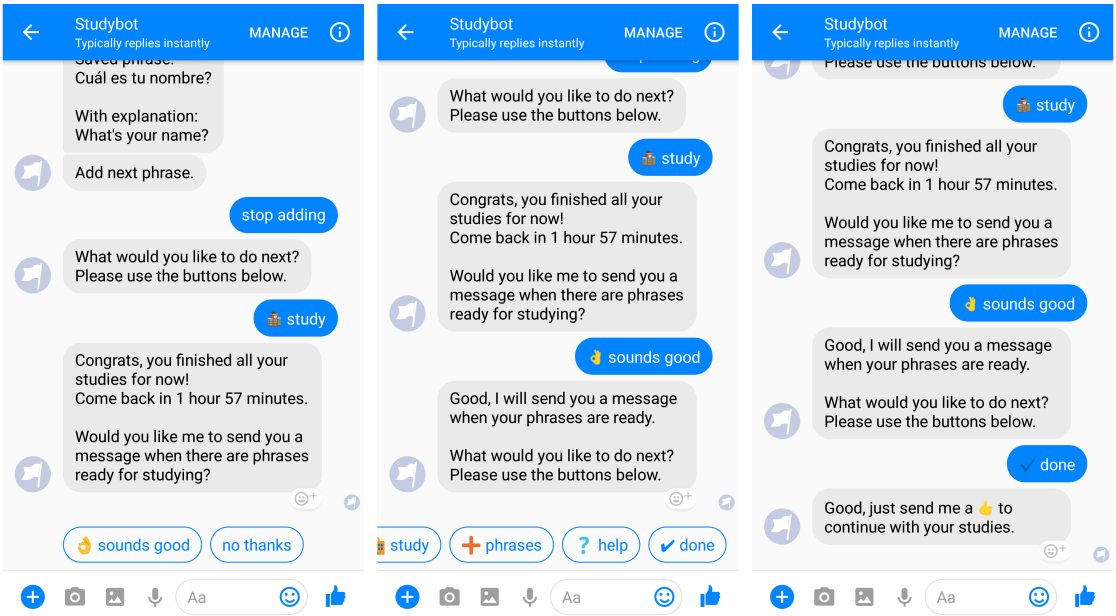

Figure 4.6: Attempt to study and activation of notifications

By clicking on the button labeled **study**, the user switches to the *mode* for studying of the added vocabulary.
However, as the image on the left of figure 4.6 shows, the added phrases cannot be studied yet.

Studybot uses a concept known as *spaced repetition system*, or short *SRS*, which makes use of the so called *spacing effect*. "Information that is spaced over time is better remembered than the same amount of information massed together" [46].
Based on this approach users need to wait before studying newly added vocabulary. The more often a phrase is guessed correctly the less frequent the user will be asked to review the phrase.

Since added phrases can not be studied directly, the user is instead offered to receive a notification message once phrases are ready for review.

When sending notifications to users, it is crucial to not send more messages than necessary. Depending on the messenger platform, there are also policies in place on how many messages

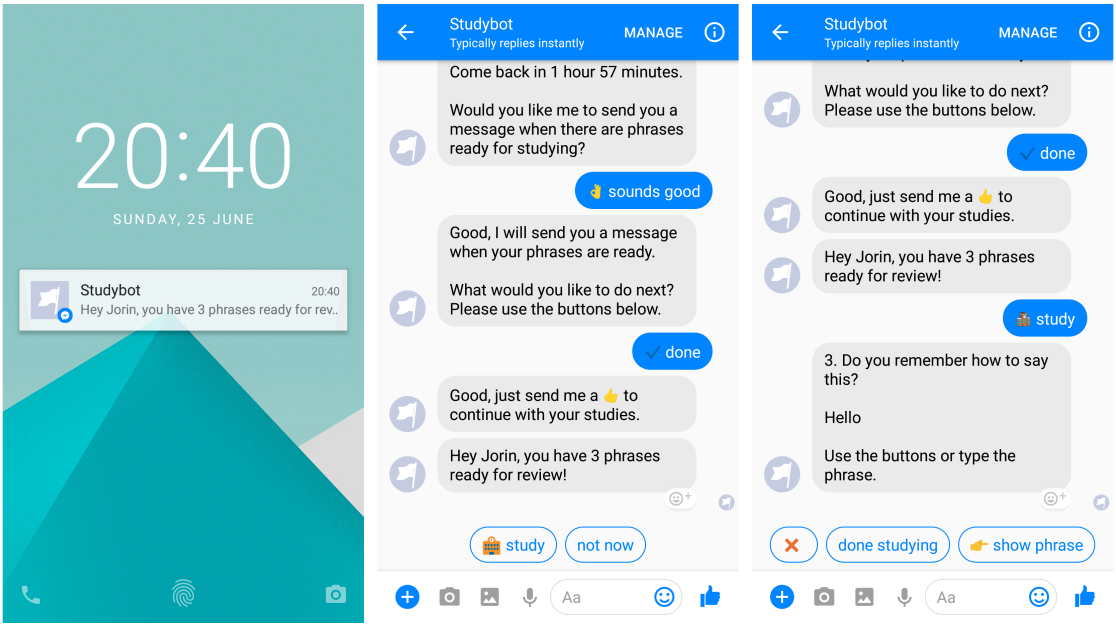

Figure 4.7: Notification message and beginning of study

are permitted.
The platform policy of *Facebook Messenger* clearly requires to "respect all requests (either on Messenger or off) by people to block, discontinue, or otherwise opt-out of your using Messenger to communicate with them" [47], which means that there needs to be a way for users to disable the sending of notifications. Further the platform policy clarifies that "you may message people within 24 hours of a person's interaction with your business or Bot ..., and until the next interaction, you may send one additional message after this 24 hour period in order to follow up on your conversation" [47].
How notifications can be disabled in *Studybot* is not further explained here, but an illustration of the feature can be found in the appendix in figure A.1 on page 50.

In figure 4.7 a notification from the *Messenger* application is shown on the mobile phone. It is triggered when enough studies are ready to be reviewed.
When opening the conversation with the chatbot, the user is now prompted to study.
As defined as a requirement in 4.3.2 on page 28, while studying one is free to choose to answer either by using the buttons at the bottom of the screen or by directly typing the correct phrase.

The **x** button available on the right image of figure 4.7, allows users to delete a phrase.
Since there is no possibility in *Facebook Messenger* to display a long list of interactive elements

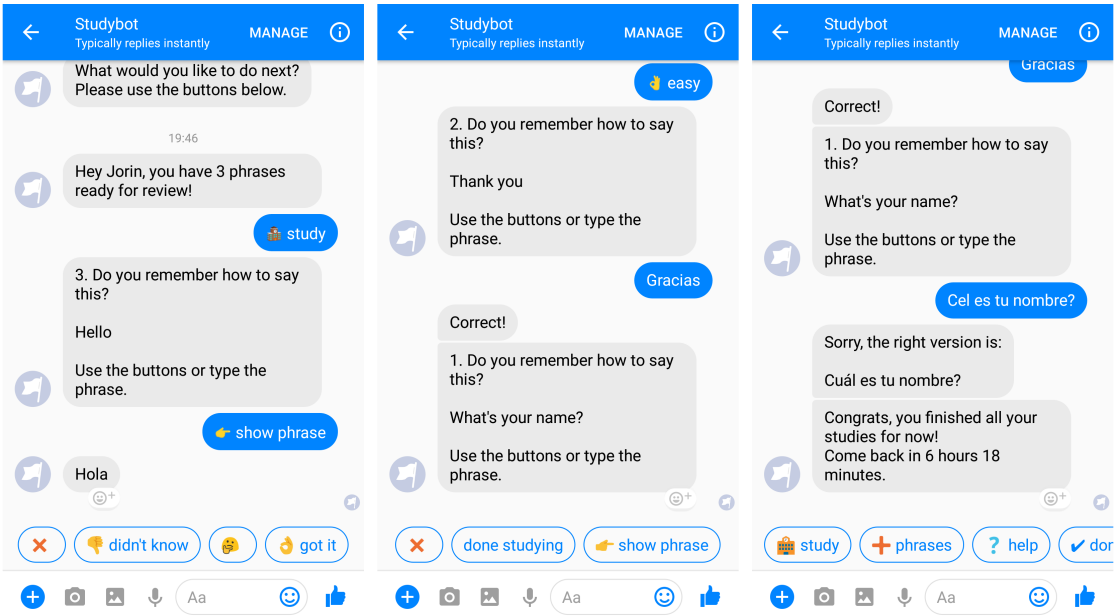

Figure 4.8: Study using buttons or by typing

to the user, creating separate functionality for editing and deleting phrases is difficult to achieve, and during studying is the only possible moment to refer to a phrase.

The figures 4.7 and 4.8 show a subtle decision to display the buttons users are most likely to interact with on the right side.
Since the English language is written from left to right and we scan text accordingly, it is more intuitive to show the most important information on the left, however certain interactions, like studying, need to be performed so frequently, that a user will remember the location of the buttons quickly and does not need to *scan* the interface every single time.
In this scenario it is helpful to have the most used action on the right side, because the majority of humans are right-handed [48], and they can therefore reach the buttons on the right with less effort since they are closer to the thumb when using a mobile phone single-handed.

A feature, that is outside of the main flow of *Studybot*, is the sending of feedback to the developers of the chatbot.
As shown in figure 4.9, it can be accessed after pressing the *help* button.
This feature is not specific to language studying and it is useful for most chatbots.
Normally all messages are answered automatically by the chatbot, but there should be a way

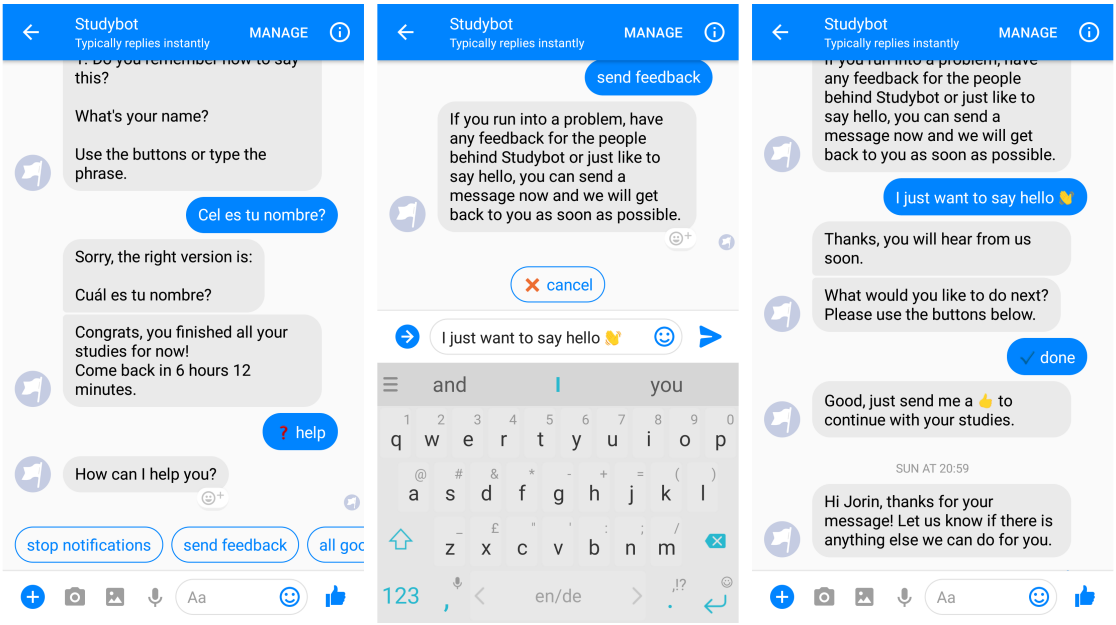

Figure 4.9: Send feedback and receive a reply

for users to talk to the human developers or administrators that provide the chatbot.
The feature is automated by sending the messages that users send as *feedback* to a *Slack channel* [49].
Additionally administrators can directly reply to user messages inside *Slack* and the replies are send back to the correct users within the chatbot.
Figure 4.9 shows the flow for sending feedback from a user's point of view.
The exact implementation is hidden from users, but simplifying the sending of replies by using an existing medium like *Slack*, can minimize the manual work required to maintain a personal feature like this.

All primary interactions the example chatbot provides have been covered above.
Many of the implemented interactions can be transfered and reused when creating other kinds of chatbots; addressing users by name, sending text in small chunks with a delay in between, prompting users for specific input with every message, keeping track of the context of each user, asking for permission before sending notifications, consciously ordering buttons and supporting custom user feedback, these patterns can be applied in many different scenarios.

The resulting functionality and implementation of the example chatbot can be summarized as finding the simplest possible interaction for the user to get a task done, which is expressed

well in the following quote from the newspaper *The Guardian* [50] about the lessons they learned by developing a chatbot:

> A lot of users responded as they would to a human, and when they got non-human responses, they'd stop using it, said Wilk. So the Guardian went in the opposite direction with its news bot and aimed for utter simplicity. The lesson, according to Wilk, was: "Don't build people's expectations too much of what's possible, just keep it simple."

4.6 Server Architecture

The following is an overview about how *Studybot* has been created.
Instead of covering all details, the focus here is to communicate the underlying concepts, which are not unique to this specific chatbot and thereby enable others to apply these patterns in their own development.

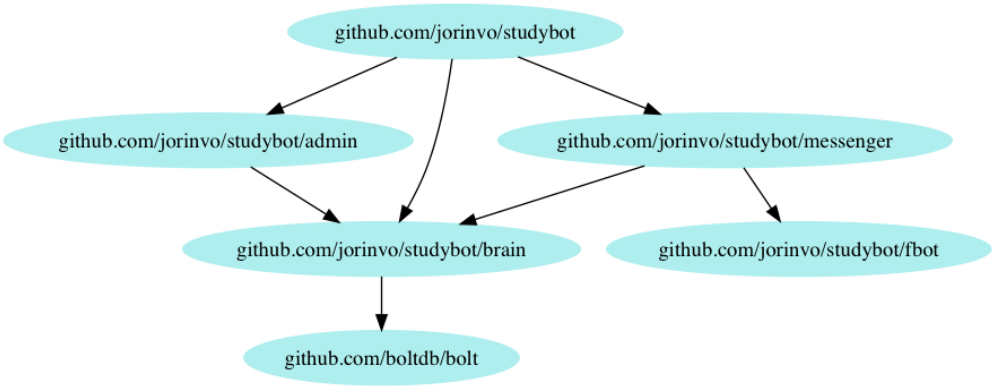

Figure 4.10: Package dependency graph[5]

There are numerous resources available today that demonstrate the development of chatbots. However a majority of existing material relies on specific services, frameworks or libraries for the implementation.
The implementation of *Studybot* demonstrates all basics necessary for the creation of a chatbot without relying on external tooling; the code-base has no external dependencies, with the exception of a package for the database.

5 The graph has been created with *godepgraph* [51]

The Graph in figure 4.10 shows the overall architecture of the chatbot.
The **main** package handles only configuration, setup and tear-down. It instantiates the *data store* from package **brain** and starts web servers listening on two separate ports for the packages **messenger** and **admin**.

The *store* has a connection to the database and it is responsible for all domain-specific business logic of *Studybot*. Both servers use the *store* to fetch data and they therefore depend on package **brain**.
Package **admin** only provides functionality for internal use by the administrators of the chatbot; one of its main responsibilities is to handle communication with *Slack* for the in 4.5 on page 40 mentioned *feedback* feature.

Package **messenger** is responsible for processing all events received from the *Facebook Messenger platform* and sending replies back to users.

It relies on another package named **fbot**, which is a simple abstraction over functionality of the *Facebook Messenger platform*. Communication in this package happens via *JSON over HTTP*.
Since this is a custom package, it is specifically designed to be used in this chatbot. It only needs to support data types and parameters that are relevant to this chatbot.

Most of the code base and its architecture are structure the same way most other *servers* are organized; the logic specific to the development of the chatbot is contained in the package **messenger**.
Figure 4.11 illustrates the behavior of this package. For brevity calls to the packages **brain** and **fbot** are not included in the graphic.
The upper half of the graphic in 4.11 shows function calls, that are invoked from package **main** for initializing an instance of the type *Bot*.

The type *Bot* provides functionality to handle events coming from the *Facebook Messenger platform* and to send generated responses back to users.
The function *HandleEvent*, which can be seen in figure 4.11, is called by package **fbot** with every event the WebHook receives, and it is the main entry-point to the chatbot logic.

6 The visualization has been created with *go-callvis* [52]

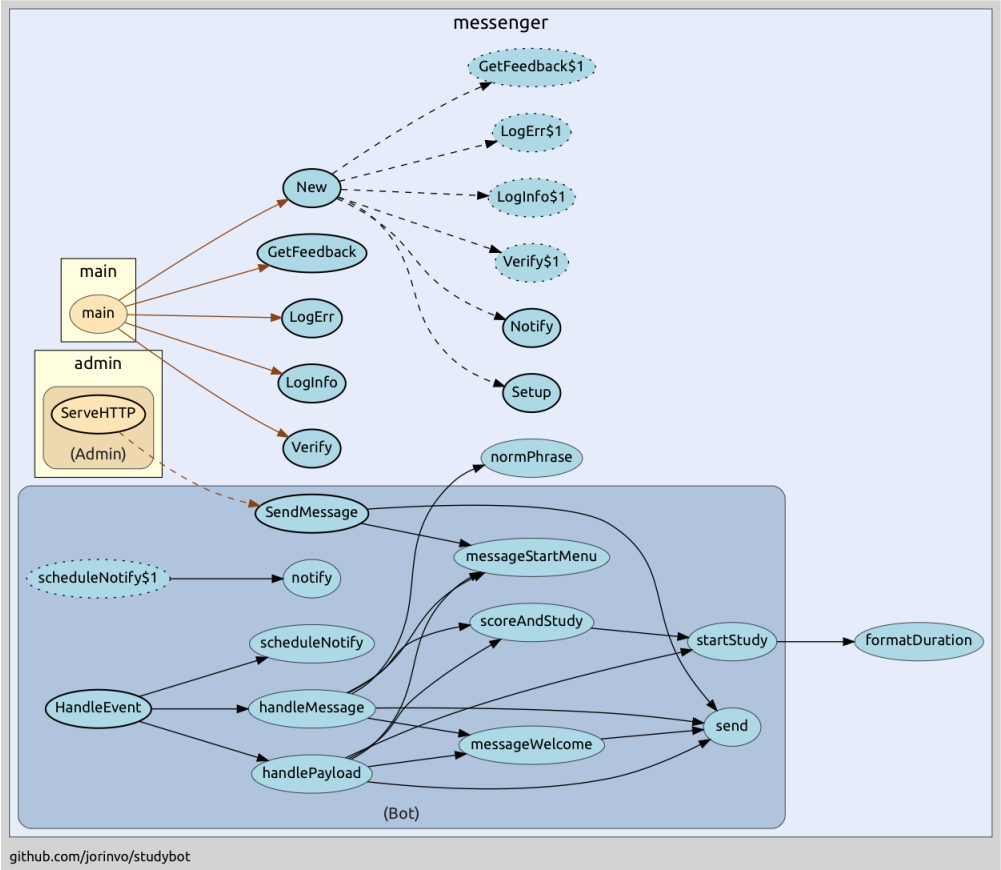

Figure 4.11: Call Graph of the messenger package[6]

Depending on the type of event, *HandleEvent* tracks whenever a user reads a message, or it sends a message back to the user, whereby it needs to be differentiated between handling text messages or predefined buttons.

In addition to directly responding to users, *HandleEvent* is also the trigger for scheduling notifications.
With every action of users, the next time they receive a notification needs to be rescheduled, which happens in the call to the internal function *scheduleNotify*. The function in 4.11 labeled as *scheduleNotify$1* is an *anonymous callback function*, one for each user, that is called by the scheduled timer to send a message to a user.

Further, it is visible in 4.11, that package **admin** can call a function named *SendMessage* which belongs to the *Bot* type. This is used to send replies from *Slack* back to users.

Separating and containing logic specific to the chosen platform, in this case *Facebook Messenger*, makes it easier to adopt the chatbot to new platforms in the future.

Call graphs for the packages **main**, **admin** and **fbot** can be found in the appendix starting on page 51. Additionally the Go documentation for all exported types and functions can be found on page 53. It explains specific implementations in more detail.

The development of a chatbot can be summarized as being a slight variation of already known server-side development.
Chatbot software is fundamentally a server accepting *HTTP requests* from a messenger platforms and sending *HTTP responses* back to the messenger platform.
Developing such a system should feel familiar to all developers that have in the past created web applications with server-side rendering, since state management and request processing work in the same manner.
While the receiving of user events can be approached with known patterns, the sending of replies back to users requires a paradigm shift in thinking.
Technically responses are also rendered and sent to users, but the rendering has no custom underlying interface as it is the case when rendering *HTML* for web pages. Instead, primarily plain text and only few platform-specific interface elements are available for presenting content.

Apart from potential use of complex natural language technology, the main difficulty of chatbot development is the design of user interfaces for the medium *chat*.

5 Future Development

To conclude the exploration of chatbots, a more opinionated take on the merits of chatbots and also ideas in which direction their role could evolve in the future are given based on thoughts from different people active in the field.

To begin with it should be clarified that text is a great medium for communication as this quote [53] illustrates:

> Text is the most socially useful communication technology. It works well in 1:1, 1:N, and M:N modes. It can be indexed and searched efficiently, even by hand. It can be translated. It can be produced and consumed at variable speeds. It is asynchronous. It can be compared, diffed, clustered, corrected, summarized and filtered algorithmically. It permits multiparty editing. It permits branching conversations, lurking, annotation, quoting, reviewing, summarizing, structured responses, exegesis, even fan fic. The breadth, scale and depth of ways people use text is unmatched by anything.

But no matter how useful text is, users prefer to not to type out everything in text. "If something can be tapped/clicked instead of typing, they prefer that" [54].

The idea of chatbots exists for a long time already, but just now messenger platforms add features that help developers to make chatbots more user-friendly by not requiring users to type out every response.
"Through our journey, we have understood that the best way to build bots for businesses is through a hybrid approach – use of buttons and quick replies along with text-based queries. This approach helps both businesses and customers," says Mounish, a co-founder of the bot builder platforms MindIQ [32]. Neither is natural language processing based on the latest advancements in machine learning and artificial intelligence a solution for every problem [55]:

> "Adding natural language for simple domains is overkill," says Dennis Thomas,

> CTO at NeuraFlash, which develops AI tools that integrate into Salesforce ... "When you have a visual medium and buttons can accomplish the task in a couple clicks (think easy re-order), open-ended natural language is not making the user's life easier."

The recent additions of capabilities to messenger platforms give chatbots the ease of use other applications provide, while chatbots can also take advantage of the flexibility and power of text as a medium.

Text and natural language enable new kinds of products, where the focus is more on flexible and individual behavior than on accuracy.
A "place where NLP is a big win is when the bot's objective is focused on helping users with the discovery phase of products or shopping." [55] Businesses selling goods, newspapers that need to spread their content and travel agencies helping customers to create individual journeys, these are some branches which are currently particularly interested in the possibilities of natural language.
Two areas the usage of chatbots is promising for are *health care* and *banking* [56] because these fields involve a big amount of customer service of which many common situations could potentially be automated with machine-based customer support.

Another interesting development, apart from enhancements of the interfaces and natural language, is the in 3.5 on page 18 mentioned category of *personal assistants*:

Many chatbots, including the created example, are designed to only solve a specific problem. They always have to find a way to handle attempts of the user to use them for tasks outside of their area of expertise. *Personal assistants* do not have these limitations and systems like *Siri* or *Google Assistant* have the potential to evolve into assistants as imagined in *science fiction* movies and literature [57].

Particularly interesting ideas are open systems where third-party developers can add additional capabilities, and the *personal assistant* has a mechanism of delegating specific tasks to the third-party systems. An example is the development of *skills* for Amazon's *Alexa* [58].

Lastly, after focusing on *chatbots* for the whole time, it can be questioned if this is the right terminology after all [59].

> First, the term "chatbot" sets an expectation around the user experience that the technology can't deliver. "Chat" is a very human word. You chat with your friends. You have chat with your neighbor. Chatting has specific connotations – it's very casual and easy. Chats meander, and you can take them any direction you want. ... If business users think "chatbots" are trivial, or if they simply prefer a fancier word to refer to the function ("business process automation") then we're setting ourselves up for hard conversations with potential business buyers.

Depending on the circumstances a different term can be more appropriate.

But no matter whether they are called *chatbots*, *bots*, *business process automation* or *conversational interfaces*, they can be used to imagine new product ideas and they bring the power and convenience of text to where users feel most at home.

6 Conclusions

This work introduced the fundamentals of what chatbots are.
It gave an overview about ideas, products and platforms, both, from the past and available today.
The current interest in chatbots, potential use cases and limitations have been explained.
Further, details of the implementation of a chatbot and the different aspects of working with conversational interfaces have been presented through the creation of an exemplary chatbot, which included interaction design for the user experience and a general, reusable software architecture for chatbots.

While not all aspects can be covered in the context of this work, the goal was to give an overview about what chatbots are, their use cases and how to create them.

This knowledge should help exploring further possibilities of chatbot usage and it should enable more developers to apply chatbots to new scenarios and thereby also improve human-machine interaction in general.

A Figures

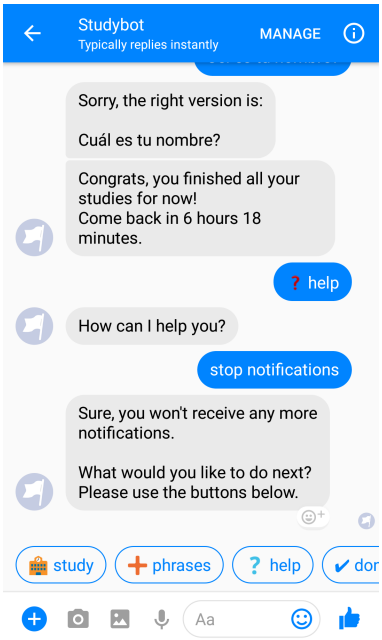

Figure A.1: Disable notifications

B Call Graphs

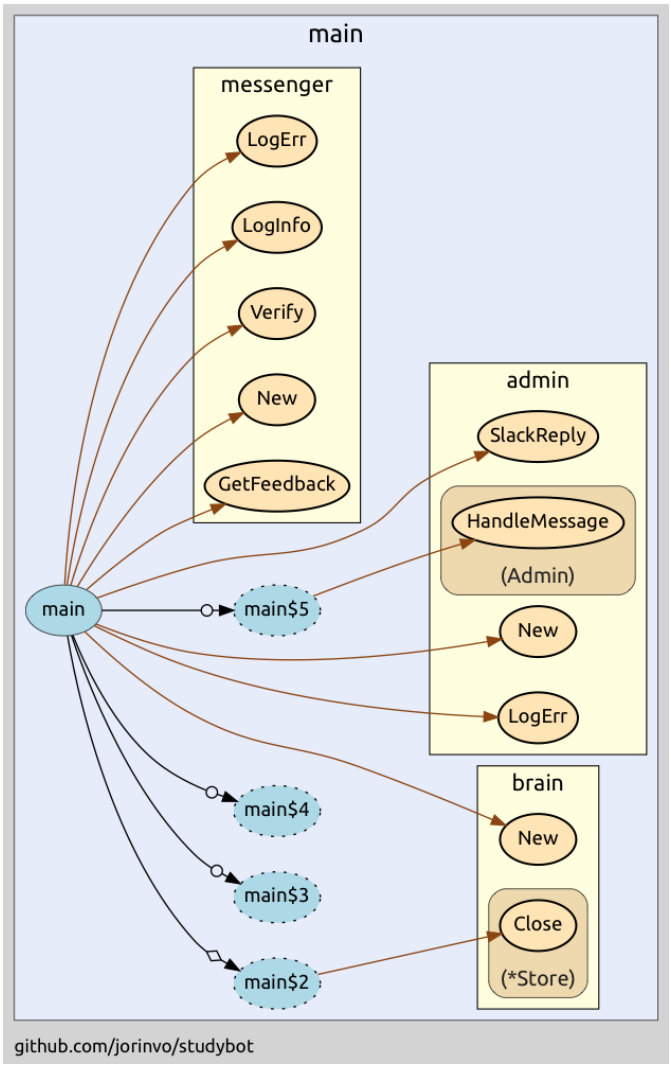

Figure B.1: Call Graph of package main

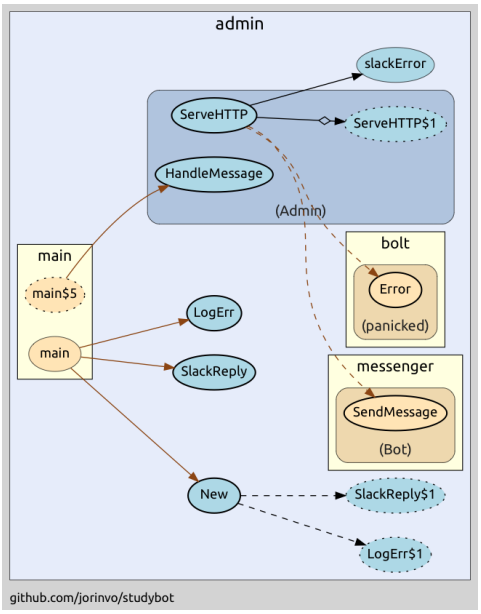

Figure B.2: Call Graph of package admin

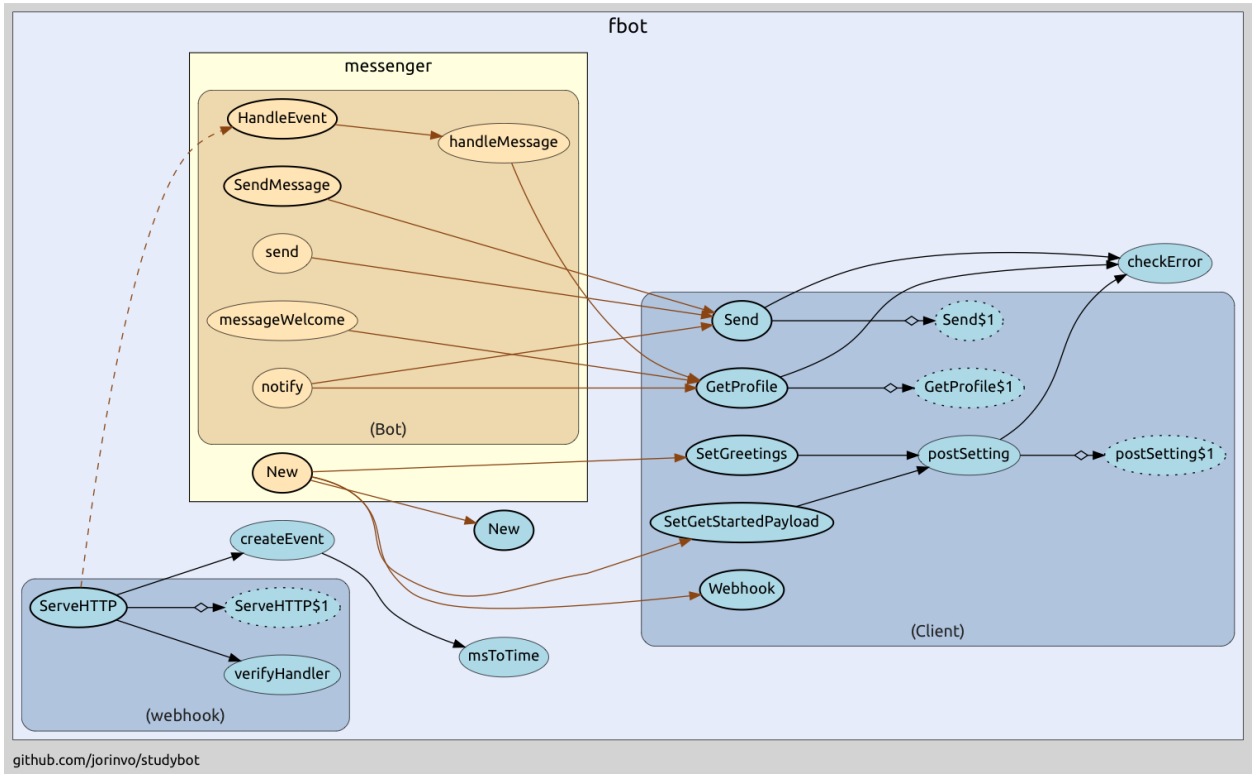

Figure B.3: Call Graph of package fbot

C Documentation

Listing C.1: Command line interface usage information

```
Studybot - Facebook Messenger bot

Usage: studybot [flags]

Studybot uses BoltDB as a database.
Data is stored in a single file. No external system is needed.
However, only one application can access the database at a time.

Studybot starts a server to serve a webhook handler that can be registered as a Messenger bot.
The server is HTTP only and a proxy server should be used to make the bot available on
a public domain, preferably HTTPS only.

An admin server runs on a separate port.
It should be proxied and secured via HTTPS + basic auth.
The admin server provides an endpoint to fetch backups of the database.
Further, it provides an endpoint that can be registered as Slack Outgoing Webhook.

When users send feedback to the bot, the messages are forwarded to Slack
and admin replies in Slack are send back to the users.


Flags:
  -admin int
        Port admin interface listens on. (default 8081)
  -db string
        Required. Path to BoltDB file. Will be created if non-existent.
  -port int
        Port Facebook webhook listens on. (default 8080)
  -slackhook string
        Required. URL of Slack Incoming Webhook. Used to send user messages to admin.
  -slacktoken string
        Token for Slack Outgoing Webhook. Used to send admin answers to user messages.
  -token string
        Required. Messenger bot token.
  -verify string
        Required. Messenger bot verify token.
```

Listing C.2: Go documentation for package messenger

```
1  package messenger
2      import "github.com/jorinvo/studybot/messenger"
3
4      Package messenger implements the Messenger bot and handles all the user
5      interaction.
6
7  FUNCTIONS
8
9  func GetFeedback(f chan<- Feedback) func(*Bot)
10     GetFeedback sets up user feedback to be sent to the given channel.
11
12 func LogErr(l *log.Logger) func(*Bot)
13     LogErr is an option to set the error logger of the bot.
14
15 func LogInfo(l *log.Logger) func(*Bot)
16     LogInfo is an option to set the info logger of the bot.
17
18 func Notify(b *Bot)
19     Notify enables sending notifications when studies are ready.
20
21 func Setup(b *Bot)
22     Setup sends greetings and the getting started message to Facebook.
23
24 func Verify(token string) func(*Bot)
25     Verify is an option to enable verification of the webhook.
26
27 TYPES
28
29 type Bot struct {
30     http.Handler
31     // contains filtered or unexported fields
32 }
33     Bot is a messenger bot handling webhook events and notifications. Use
34     New to setup and use register Bot as a http.Handler.
35
36 func New(store brain.Store, token string, options ...func(*Bot)) (Bot, error)
37     New creates a Bot. It can be used as a HTTP handler for the webhook. The
38     options Setup, LogInfo, LogErr, Notify, Verify, GetFeedback can be used.
39
40 func (b Bot) HandleEvent(e fbot.Event)
41     HandleEvent handles a Messenger event.
42
43 func (b Bot) SendMessage(id int64, msg string) error
44     SendMessage sends a message to a specific user.
45
46 type Feedback struct {
47     ChatID   int64
48     Username string
49     Message  string
50 }
51     Feedback describes a message from a user a human has to react to
```

Listing C.3: Go documentation for package admin

```
1  package admin
2      import "github.com/jorinvo/studybot/admin"
3
4      Package admin provides an admin server that can be used to make
           backups
5      and to communicate with users via Slack.
6
7  FUNCTIONS
8
9  func LogErr(l *log.Logger) func(*Admin)
10     LogErr is an option to set the error logger.
11
12 func SlackReply(token string, fn func(int64, string) error) func(*Admin)
13     SlackReply isa n option to enable /slack to receive replies from Slack
           .
14     token is used to validate posts to the webhook. fn is called with a
15     chatID and a message.
16
17 TYPES
18
19 type Admin struct {
20     // contains filtered or unexported fields
21 }
22     Admin is a HTTP handler that can be used for backups and to
           communicate
23     with users via Slack.
24
25 func New(store brain.Store, slackHook string, options ...func(*Admin))
       Admin
26     New returns a new Admin which can be used as an http.Handler.
27
28 func (a Admin) HandleMessage(id int64, name, msg string)
29     HandleMessage can be called to send a user message to Slack.
30
31 func (a Admin) ServeHTTP(w http.ResponseWriter, r *http.Request)
32     ServeHTTP serves the different endpoints the admin server provides.
```

Listing C.4: Go documentation for package fbot

```
1  package fbot
2      import "github.com/jorinvo/studybot/fbot"
3
4      Package fbot can be used to communicate with a Facebook Messenger bot.
5      The supported API is limited to only the required use cases and the data
6      format is abstracted accordingly.
7
8  FUNCTIONS
9
10 func API(url string) func(*Client)
11     API can be passed to New for sending requests to a different URL. Must
12     not contain trailing slash.
13
14 TYPES
15
16 type Button struct {
17     // Text is the text on the button visible to the user
18     Text string
19     // Payload is a string to identify the quick reply event internally in your application.
20     Payload string
21 }
22     Button describes a text quick reply.
23
24 type Client struct {
25     // contains filtered or unexported fields
26 }
27     Client can be used to communicate with a Messenger bot.
28
29 func New(token string, options ...func(*Client)) Client
30     New rerturns a new client with credentials set up.
31
32 func (c Client) GetProfile(id int64) (Profile, error)
33     GetProfile fetches a user profile for an ID.
34
35 func (c Client) Send(id int64, message string, buttons []Button) error
36     Send a text message with a set of quick reply buttons to a user.
37
38 func (c Client) SetGetStartedPayload(p string) error
39     SetGetStartedPayload displays a "Get Started" button for new users. When
40     a users pushes the button, a postback with the given payload is
41     triggered.
42
43 func (c Client) SetGreetings(greetings map[string]string) error
44     SetGreetings sets the text displayed in the bot description. Pass a map
45     of locale to greeting text. Include "default" locale as fallback for
46     missing locales.
47
48 func (c Client) Webhook(handler func(Event), verifyToken string) http.Handler
49     Webhook returns a handler for HTTP requests that can be registered with
50     Facebook. The passed event handler will be called with all received
51     events.
```

```
52
53 type Event struct {
54     // Type helps to decide how to react to an event.
55     Type EventType
56     // ChatID identifies the user. It's a Facebook user ID.
57     ChatID int64
58     // Time describes when the event occured.
59     Time time.Time
60     // Text is a message a user send for EventMessage and and error description for
          EventError.
61     Text string
62     // Payload is a predefined payload for a quick reply or postback sent with EventPayload.
63     Payload string
64 }
65     Event contains information about a user action.
66
67 type EventType int
68     EventType helps to distinguish the different type of events.
69
70 const (
71     // EventUnknown is the default and will be used if none of the other types match.
72     EventUnknown EventType = iota
73     // EventMessage is triggered when a user sends Text, stickers or other content.
74     // Only text is available at the moment.
75     EventMessage
76     // EventPayload is triggered when a quickReply or postback Payload is sent.
77     EventPayload
78     // EventRead is triggered when a user read a message.
79     EventRead
80     // EventError is triggered when the webhook is called with invalid JSON content.
81     EventError
82 )
83
84 type Profile struct {
85     Name     string  `json:"first_name"`
86     Locale   string  `json:"locale"`
87     Timezone float64 `json:"timezone"`
88 }
89     Profile has all public user information we need; needs to be in sync
90     with the URL abouve.
```

Listing C.5: Go documentation for package brain

```
1  package brain
2      import "github.com/jorinvo/studybot/brain"
3
4      Package brain handles all business logic of Slangbrain. It handles data
5      storage, retrieving and updating. It's independent from the used bot
6      platform and user interaction.
7
8  TYPES
9
10 type Mode int
11     Mode is the state of a chat. We need to keep track of the state each
12     chat is in.
13
14 const (
15     // ModeMenu shows the main menu.
16     ModeMenu Mode = iota
17     // ModeAdd lets the user add new phrases.
18     ModeAdd
19     // ModeStudy goes to phrases ready to study.
20     ModeStudy
21     // ModeGetStarted sends an introduction to the user.
22     ModeGetStarted
23     // ModeFeedback allows the user to send a message that is ready by a human.
24     ModeFeedback
25 )
26
27 type Phrase struct {
28     Phrase      string
29     Explanation string
30     Score       int
31 }
32     Phrase describes a phrase the user saved.
33
34 type Store struct {
35     // contains filtered or unexported fields
36 }
37     Store provides functions to interact with the underlying database.
38
39 func New(dbFile string) (Store, error)
40     New returns a new Store with a database already setup.
41
42 func (store Store) AddPhrase(chatID int64, phrase, explanation string) error
43     AddPhrase stores a new phrase.
44
45 func (store Store) BackupTo(w http.ResponseWriter)
46     BackupTo streams backup as an HTTP response.
47
48 func (store Store) Close() error
49     Close the underlying database connection.
50
51 func (store Store) DeleteChat(chatID int64) error
```

```
52      DeleteChat removes all records of a given chat.
53
54  func (store Store) DeletePhrases(fn func(int64, Phrase) bool) (int, error)
55      DeletePhrases removes all phrases fn matches.
56
57  func (store Store) DeleteStudyPhrase(chatID int64) error
58      DeleteStudyPhrase deletes the phrase the passed user currently has to
59      study.
60
61  func (store Store) EachActiveChat(fn func(int64)) error
62      EachActiveChat runs a function for each chat where the user has been
63      active since the last notification has been sent.
64
65  func (store Store) FindPhrase(chatID int64, fn func(Phrase) bool) (Phrase, error)
66      FindPhrase returns a phrase belonging to the passed user that matches
67      the passed function.
68
69  func (store Store) GetChatIDs() ([]int64, error)
70      GetChatIDs returns chatIDs of all users.
71
72  func (store Store) GetMode(chatID int64) (Mode, error)
73      GetMode fetches the mode for a chat.
74
75  func (store Store) GetNotifyTime(chatID int64) (time.Duration, int, error)
76      GetNotifyTime gets the time until the user should be notified to study.
77      Returns the time until the next studies are ready and a count of the
78      ready studies. The returned duration is always at least dueMinInactive.
79      The count is 0 if the chat has no phrases yet.
80
81  func (store Store) GetPhrasesAsJSON(chatID int64) (io.Reader, error)
82      GetPhrasesAsJSON ...
83
84  func (store Store) GetStudy(chatID int64) (Study, error)
85      GetStudy returns the current study the user needs to do.
86
87  func (store Store) IsSubscribed(chatID int64) (bool, error)
88      IsSubscribed checks if a user has notifications enabled.
89
90  func (store Store) ScoreStudy(chatID int64, score int) error
91      ScoreStudy sets the score of the current study and moves to the next
92      study.
93
94  func (store Store) SetActivity(chatID int64, t time.Time) error
95      SetActivity sets the last time a message was sent to a user.
96
97  func (store Store) SetMode(chatID int64, mode Mode) error
98      SetMode updates the mode for a chat.
99
100 func (store Store) SetRead(chatID int64, t time.Time) error
101     SetRead sets the last time the user read a message.
102
103 func (store Store) StudyNow() error
```

```
104     StudyNow resets all study times of all users to now.
105
106 func (store Store) Subscribe(chatID int64) error
107     Subscribe enables notifications for a user.
108
109 func (store Store) Unsubscribe(chatID int64) error
110     Unsubscribe disables notifications for a user.
111
112 type Study struct {
113     // Phrase is the phrase the user needs to guess.
114     Phrase string
115     // Explanation is the explanation displayed to the user.
116     Explanation string
117     // Total is the total number of studies ready, including the current one.
118     Total int
119     // Next contains the time until the next study is available;
120     // it's only set if Total is 0.
121     Next time.Duration
122 }
123     Study is a study the current study the user needs to answer.
```

Bibliography

[1] *Chat - Definition of chat in English.* URL: https://en.oxforddictionaries.com/definition/chat (visited on 04/21/2017).

[2] *Conversation - Definition of conversation in English.* URL: https://en.oxforddictionaries.com/definition/conversation (visited on 04/21/2017).

[3] *Bot - Definition of bot in English.* URL: https://en.oxforddictionaries.com/definition/bot (visited on 04/21/2017).

[4] *Interface - Definition of interface in English.* URL: https://en.oxforddictionaries.com/definition/interface (visited on 04/25/2017).

[5] *Command - Definition of command in English.* URL: https://en.oxforddictionaries.com/definition/command (visited on 04/21/2017).

[6] Alan Turing. *Computing Machinery and Intelligence.* Mind, 1950. DOI: 10.1093/mind/LIX.236.433.

[7] Carl R Rogers. *A way of being.* Houghton Mifflin Co., 1995. ISBN: 0395755301.

[8] Güven Güzeldere and Stefano Franchi. "dialogues with colorful personalities of early ai". In: *Stanford Humanities Review* 4.2 (July 24, 1995). URL: https://web.stanford.edu/group/SHR/4-2/text/toc.html (visited on 07/25/2017).

[9] Joseph Weizenbaum. *Computer power and human reason: from judgment to calculation.* W. H. Freeman, 1976.

[10] Adam Curtis. *Adam Curtis - NOW THEN.* July 25, 2014. URL: http://www.bbc.co.uk/blogs/adamcurtis/entries/78691781-c9b7-30a0-9a0a-3ff76e8bfe58 (visited on 05/03/2017).

[11] *Internet History of 1970s.* URL: http://www.computerhistory.org/internethistory/1970s/ (visited on 05/03/2017).

[12] *About the Jabberwacky AI.* URL: http://www.jabberwacky.com/j2about (visited on 05/03/2017).

[13] "Soundblaster". In: *PC Mag* 10.18 (1991), p. 67. ISSN: 0888-8507. URL: https://books.google.co.th/books?id=fpQP3e54P-gC.

[14] Clive Thompson. *Approximating Life*. July 7, 2002. URL: http://www.nytimes.com/2002/07/07/magazine/approximating-life.html (visited on 05/03/2017).

[15] *Twelfth National Conference on Artificial Intelligence*. URL: http://www.aaai.org:80/Library/AAAI/aaai94contents.php (visited on 05/03/2017).

[16] *Apple Launches iPhone 4S, iOS 5 & iCloud*. URL: https://www.apple.com/pr/library/2011/10/04Apple-Launches-iPhone-4S-iOS-5-iCloud.html (visited on 05/03/2017).

[17] Roger Parloff. *The AI Revolution: Why Deep Learning Is Suddenly Changing Your Life*. Sept. 28, 2016. URL: http://fortune.com/ai-artificial-intelligence-deep-machine-learning/ (visited on 05/03/2017).

[18] James Titcomb. *Mobile web usage overtakes desktop for first time*. Nov. 1, 2016. URL: http://www.telegraph.co.uk/technology/2016/11/01/mobile-web-usage-overtakes-desktop-for-first-time/ (visited on 05/03/2017).

[19] Eve-Marie Lanza. *cn i TLK 2 u? How to capitalize on emerging conversation trends*. Dec. 15, 2016. URL: https://www.ibm.com/blogs/bluemix/2016/12/capitalize-emerging-conversation-trends/ (visited on 05/03/2017).

[20] Tomi T Ahonen. *Time to Confirm some Mobile User Numbers: SMS, MMS, Mobile Internet, M-News*. Jan. 13, 2011. URL: http://communities-dominate.blogs.com/brands/2011/01/time-to-confirm-some-mobile-user-numbers-sms-mms-mobile-internet-m-news.html (visited on 05/01/2017).

[21] *Most popular global mobile messenger apps as of January 2017, based on number of monthly active users (in millions)*. URL: https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/ (visited on 05/02/2017).

[22] *Statistics and facts about mobile messenger app usage*. URL: https://www.statista.com/topics/1523/mobile-messenger-apps/ (visited on 05/03/2017).

[23] Matt Mayer. *Building a WeChat (Weixin) robot*. Mar. 14, 2014. URL: https://blog.reigndesign.com/blog/building-a-wechat-weixin-robot/ (visited on 05/03/2017).

[24] Daniel Sevitt. *The Most Popular Messaging Apps by Country*. Feb. 27, 2017. URL: https://www.similarweb.com/blog/popular-messaging-apps-by-country (visited on 06/26/2017).

[25] *All API.AI Integrations - API.AI*. URL: https://docs.api.ai/docs/integrations (visited on 06/17/2017).

[26] Dotan Elharrar. *7 Types of Bots*. Jan. 10, 2017. URL: https://chatbotsmagazine.com/7-types-of-bots-8e1846535698 (visited on 05/04/2017).

[27] Mazen Abu Tawileh. *How Instant Translator bot is used by more than 50k users daily on Facebook & Viber and has more than 600k active users monthly?* Apr. 6, 2017. URL: https://botpublication.com/how-instant-translator-bot-is-used-by-more-than-50k-users-daily-on-facebook-viber-and-has-more-c42fc80a6ea3?gi=2cebdf542597 (visited on 05/04/2017).

[28] *KLM on Messenger*. URL: https://social.klm.com/flightinfo/messenger/ (visited on 07/26/2017).

[29] Ryan Kelly. *How our dumb bot attracted 1 million users without even trying*. Nov. 7, 2016. URL: https://venturebeat.com/2016/11/07/how-our-dumb-bot-attracted-1-million-users-without-even-trying/ (visited on 05/04/2017).

[30] Jon Bruner. *Joshua Browder on bots that fight bureaucracy*. Nov. 15, 2016. URL: https://www.oreilly.com/ideas/joshua-browder-on-bots-that-fight-bureaucracy (visited on 05/04/2017).

[31] Adelyn Zhou. *100 Best Bots For Brands & Businesses*. Mar. 15, 2017. URL: http://www.topbots.com/100-best-bots-brands-businesses/ (visited on 05/04/2017).

[32] Durba Ghosh. *Chatbots make a lot of noise but remain on the sidelines*. Dec. 31, 2016. URL: https://www.techinasia.com/chatbots-and-startups (visited on 05/05/2017).

[33] Michael Yuan. *The Rise of Intelligent Bots*. Apr. 26, 2016. URL: https://chatbotbook.com/the-rise-of-intelligent-bots-e896cde7281b (visited on 05/05/2017).

[34] BI Intelligence. *Messaging apps are now bigger than social networks*. Sept. 20, 2016. URL: http://www.businessinsider.com/the-messaging-app-report-2015-11?IR=T%5C&r=US%5C&IR=T. (visited on 05/05/2017).

[35] Shaul Olmert. *Chatbots in Asia: these days, everyone is getting chatty*. Feb. 7, 2017. URL: http://www.thedrum.com/opinion/2017/02/07/chatbots-asia-these-days-everyone-s-getting-chatty (visited on 05/05/2017).

[36] Alan Henry. *Five Best Language Learning Tools*. Oct. 20, 2013. URL: http://lifehacker.com/five-best-language-learning-tools-1448103513 (visited on 05/09/2017).

[37] Joseph Mapue. *Conversational Bots Transform How You Learn Languages*. Nov. 2, 2016. URL: http://www.topbots.com/duolingo-chatbots-app-change-way-you-learn-languages/ (visited on 05/09/2017).

[38] *Gamification - Definition of gamification in English*. URL: https://en.oxforddictionaries.com/definition/gamification (visited on 05/10/2017).

[39] *SMS Pricing for Text Messaging*. URL: https://www.twilio.com/sms/pricing/de (visited on 06/24/2017).

[40] *Web Hooks*. URL: https://webhooks.pbworks.com/w/page/13385124/FrontPage (visited on 06/24/2017).

[41] *asyncio - Asynchronous I/O, event loop, coroutines and tasks*. URL: https://docs.python.org/3/library/asyncio.html (visited on 06/24/2017).

[42] *Node.js*. URL: https://nodejs.org/en/ (visited on 06/24/2017).

[43] *The Go Programming Language*. URL: https://golang.org/ (visited on 06/24/2017).

[44] *boltdb/bolt: An embedded key/value database for Go.* URL: https://github.com/boltdb/bolt/ (visited on 06/24/2017).

[45] *Unicode Emoji*. URL: https://github.com/boltdb/bolt/ (visited on 06/27/2017).

[46] Tracey J. Shors Helene M. Sisti Arnold L. Glass. "Neurogenesis and the spacing effect: Learning over time enhances memory and the survival of new neurons". In: *Learning & Memory* 14.5 (May 2007), pp. 368–375. DOI: 10.1101/lm.488707.

[47] *Platform Policy - Facebook for Developers: 16. Messenger Platform*. URL: https://developers.facebook.com/policy#messengerplatform (visited on 06/27/2017).

[48] Jason G Goldman. *Evolution: Why are most of us right-handed?* Dec. 16, 2014. URL: http://www.bbc.com/future/story/20141215-why-are-most-of-us-right-handed (visited on 07/25/2017).

[49] *Slack: Where work happens*. URL: https://slack.com/ (visited on 06/28/2017).

[50] Jessica Davies. *What The Guardian has learned from chatbots*. Sept. 5, 2016. URL: https://digiday.com/uk/guardian-learned-chatbots/ (visited on 07/07/2017).

[51] *kisielk/godepgraph: A Go dependency graph visualization tool*. URL: https://github.com/kisielk/godepgraph (visited on 06/28/2017).

[52] *TrueFurby/go-callvis: Visualize call graph of your Go program using dot format*. URL: https://github.com/TrueFurby/go-callvis (visited on 06/28/2017).

[53] Jonathan Libov. *Futures of text*. Feb. 23, 2015. URL: https://whoo.ps/2015/02/23/futures-of-text (visited on 07/10/2017).

[54] Gijo Varghese. *Dear Bot developers, Don't get over hyped*. Jan. 25, 2017. URL: https://chatbotslife.com/dear-bot-developers-dont-get-over-hyped-374572412fda (visited on 07/10/2017).

[55] Paul Boutin. *Does a Bot Need Natural Language Processing?* Apr. 3, 2017. URL: https://chatbotsmagazine.com/does-a-bot-need-natural-language-processing-c2f76ab7ef11 (visited on 07/10/2017).

[56] *These 2 Industries Are The Next Big Things For Chatbots*. URL: https://blog.botlist.co/these-2-industries-are-the-next-big-things-for-chatbots/ (visited on 07/10/2017).

[57] Farhad Manjoo. *A High-Stakes Bet: Turning Google Assistant Into a 'Star Trek' Computer*. Sept. 28, 2016. URL: https://www.nytimes.com/2016/09/29/technology/google-assistant.html (visited on 07/10/2017).

[58] *Alexa Skills Kit - Build for Voice with Amazon*. URL: https://developer.amazon.com/alexa-skills-kit (visited on 07/10/2017).

[59] *Chatbots: a Misleading Term We Should Stop Using*. URL: http://botnerds.com/chatbots/ (visited on 07/10/2017).

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Ort, Datum Unterschrift