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 29, 2017

Contents

1 Fundamentals

Before working on a topic it is necessary to have common definitions of its vocabulary. This chapter will help aligning previous assumptions.

1.1 Definition of the Term Chatbot

The term *chatbot* consists of two parts - *chat* and *bot*. The meaning can be better understood after 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 *chats* and therefore *chatbots*. But before examining what characterizes a conversation other noteworthy aspects of this definition are the inherent *informal* format of a *chat*, and the traits of being *online* and *real time*.
Informality doesn't have to be seen as a strict requirement; however a chat massage and, for example, a classical letter have different degrees of formality.
Being *online* and thereby not bound to a specific location, device or other physicality can be seen as critical foundation for determining potential types of systems to interact with such media.
The aspect of limiting communication to *real time* gives an important restriction on possible interactions and sets a baseline for the expected user experience. This also excludes the usage of certain technologies that don't allow for 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].
Important here 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*. It's important to internalize that the creation of a chatbot is eventually 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* being *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 programs 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.

1.2 About 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 a means of interaction while discouraging the idea of using an interface consisting purely of static or graphical elements.
Picturing a *conversational* interface one might expect a flow of interactions consisting of elements from all involved parties similar to the interfaces humans use to interact with each other.
It should also be noted that conversations are not limited to written language. Spoken language 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 often used to interact with computers are one example of text-based interfaces which don't 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 underlaying ideas communicated opposed to merely receiving the characters the words consist of.

This deeper understanding of the intends a user has and the ability to adjust the interaction with a conversation consisting of customized parts sets conversational interfaces, and thus chatbots, apart from other interfaces.

2 Applications

The following chapters explore different applications of chatbots.
Starting with past work in the first chapter and looking at the present day in the subsequent chapters, different approaches and products are presented. Apart from highlighting interesting products, whole categories with potential benefits of using chatbots are defined and equally, problematic areas are identified as well.
Furthermore the current ecosystem is portrayed; this includes involved companies, available target platforms and additional tooling.
Moreover, current and potential users of chatbot products are analyzed and important questions about the current state of chatbot usage are addressed.
Lastly the aspired goals, potentials and promises of chatbots are further identified.

2.1 Past Work

Before working on new projects with nowadays technology one should know about prior work and learn from past ideas - both, succeed and failed attempts.
This chapter presents selected events from the last century, which introduced ideas forming the present definition of chatbots.
It should be noted that this is no attempt at presenting an all-encompassing overview about the history of computing, instead the aim is to explain where the idea of chatbots and the interest of creating them originated from.

2.1.1 The Turing Test

Even before the term *chatbot* was coined people started working on machines that interact with humans.
One of the first important milestones was the 1950 paper "Computing Machinery and Intelligence" by Alan Turing[6]. His ideas back then are still an important building block of what is call a *chatbot* in todays 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". This test is meant to identify if a machine posses human-like intelligence.
Originally Turing described this test as "imitation game" whereas the experiment consist of a human interacting with two parties via *textual messages*. One of the parties is another human and one is a machine. The human doesn't know upfront which one is the machine, but only that one of them will be a machine. During the *game* the human can interact, via what we nowadays call *chatting*, with the other party and 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. Up to this point in time the Turing Test still remains to be challenged by new systems.

2.1.2 ELIZA

14 Years after the Turing Test was defined Joseph Weizenbaum started working on what would be known as the first program to pass a (limited) version of the Turing Test. Joseph Weizenbaum began working at MIT Artificial Intelligence Laboratory in 1964 and he released the ELIZA program in 1966.

The original version of ELIZA was written in a programming programming language called MAD-Slip which was also created by Joseph Weizenbaum himself and it ran on the IBM 704 computer.
ELIZA creates responses to natural language messages 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. 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 a non-obvious question.

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 clients answer “Well, my boyfriend made me come here.” can be transformed to “Your boyfriend made you come here?”[7].
Certain signal words or 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[7].

Knowing about the nature of Rogerian psychotherapy, Joseph Weizenbaum created ELIZA initially intended as a parody showing off the simple behavior necessary to imitate this therapy.
He was surprised that even people knowing about the inner workings of the program ended up having serious conversations with ELIZA. In one anecdote Joseph Weizenbaum tells how his secretary, after starting a conversation with ELIZA, asked him: “Would you mind leaving the room, please?”[8, p. 5].

Lead by the success of the experiment he published the book "Computer power and human reason: from judgment to calculation" in 1976 which discusses 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.”[9]. This idea coined the term *Eliza Effect* which describes people assuming computers to behave like humans. This term is still in use today.

2.1.3 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 and it also models a personality including concepts of how to have 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.[10]
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

publics attention include
Jabberwacky which was started in 1988 and attempts to learn from the users input[11],
Dr. Sbaitso which was released in 1991 as an ELIZA-like demonstration for a sound card and was one of the first chatbots for MS-DOS based personal computers[12],
and A.L.I.C.E., which has been first released in 1995 and became famous for its realistic behavior that is based on heuristic patterns instead of static rules[13].

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 version *ChatterBots*.[14]

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 a real human.
It needs to be noted that although creating an as human-like as possible system remains a popular challenge, not all applications of chatbots profit 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.

2.2 Present Day

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

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

Since Apple's release of Siri in 2011[15] customers have become more aware of the possibilities 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.

Simultaneously, in this period of time artificial intelligence gained new traction due to the success of using artificial neural networks for machine learning[16].
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 now. A second important condition is the amount of data available today; big Internet companies specialize on collecting data, originally intended to better target advertisement, but now they can use their data to train neural networks. Neural networks have their name because they are modeled after human brains; instead of specific rules of what the program should do the machine learns from examples in similar ways to how humans learn. Some task that are too complex to solve with a rule-based program, can now be solved by collecting enough example data and letting the machine figure out the solution instead.
This techniques can also be applied in the field of natural language processing, which is essential to understanding and generating text for conversational interfaces.

With these 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.

In addition to these new technical possibilities 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]. This development means that users don't have space for complex interfaces on the small screens of their devices, they need a solution which is light-weight in data consumption, and they can not use complicated keyboard shortcuts. At the same time users already spend the majority of their time in messaging applications and therefore, they are well accustomed to this way of communicating.

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

2.3 Communication

Nowadays there are two fundamental ways a chatbot can communicate with users; some platforms provide user interface elements, such as buttons, that can be used; 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 something similar, the interaction can be more clear for the user and thereby simpler.
It should be noted that even by limiting interaction to predefined actions, the main characteristics of a chatbot remain; the interaction is still structure in the form of a conversation.

In certain scenarios there are too many possible user inputs to fit them in a list. Then the input has to be given as natural language.
Parsing custom input can be helpful to extract information from complex user inputs. As an example, when the user is asked for a date and time but the situation also allows for repeated times, a user could state something like "every second Tuesday at 6am and 9pm", while in a traditional user interface that would require a non-trivial amount of input fields.
If natural language is used for communication, it should be clearly stated what kind of input is expected since the user is not able to know which topics and variations of input a system understands.
One of the main challenges with a natural language interface is handling the non-restricted interacted in a coherent way; since user input is in no way limited to a certain for of expected input for the current topic, all sorts of unexpected user reactions have to be taken care of.
As a consequence of this, there are certain conversations every chatbot has to be able to handle in some way. These include, but are 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 still able to use custom 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 use any possible custom input.

2.4 Platforms

Unless software is distributed on 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 called *server*.

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

There are numerous platforms available that fulfill these requirements.

It should be noted that 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 does communicate with a server on the users 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, their capabilities and their area of focus.

To begin with, one of the most used online communication platforms is E-mail.
E-mail however does not provide the earlier in 1.1 defined characteristics of chatbots to be able to communicate *informally* and in *real time*; which disqualify E-mail as a platform for chatbots, even though in practice the use cases of chatbots overlap with the ones that can be solved with 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 give. Still it is indeed possible for this 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*, short SMS.
SMS is primarily used on mobile devices and users are identified by their phone numbers, wherefore the communication has to happen through cell network providers. 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] people worldwide which means it remains one of the most popular communication channels; and therefore is an interesting option for applications requiring a low entry barrier.

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

Apple's *Siri* is another voice based system available; but 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 task and in both cases they allow delegating specific tasks to third parties so they can handle the task at hand.

Currently popular target platforms for chatbots are messenger platforms. They are primarily text based, they mostly come without cost for the end user 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 to the user, which can help guiding users instead of exclusively relying on natural language for communication.
At this point it is not feasible to create a comprehensive list of available features for each platform, since the space 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[21] one of the most popular messenger applications. It, however, does 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[21] as well. Contrary to WhatsApp, Facebook Messenger provides a platform for developing chatbots. Counting over 100,000 monthly active chatbots[22] it is an interesting platforms to develop for.

Following in popularity[23] are two Chinese messenger platforms, *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[24].

Further popular messenger applications are the Japanese *Line* Microsoft's *Skype Telegram* and the more business focused *Slack* All of these applications also provide dedicated platforms for the development of chatbots.

The choice of platform mostly depends on the target market. Different audiences prefer different platforms and a product might be better suited for certain environments.

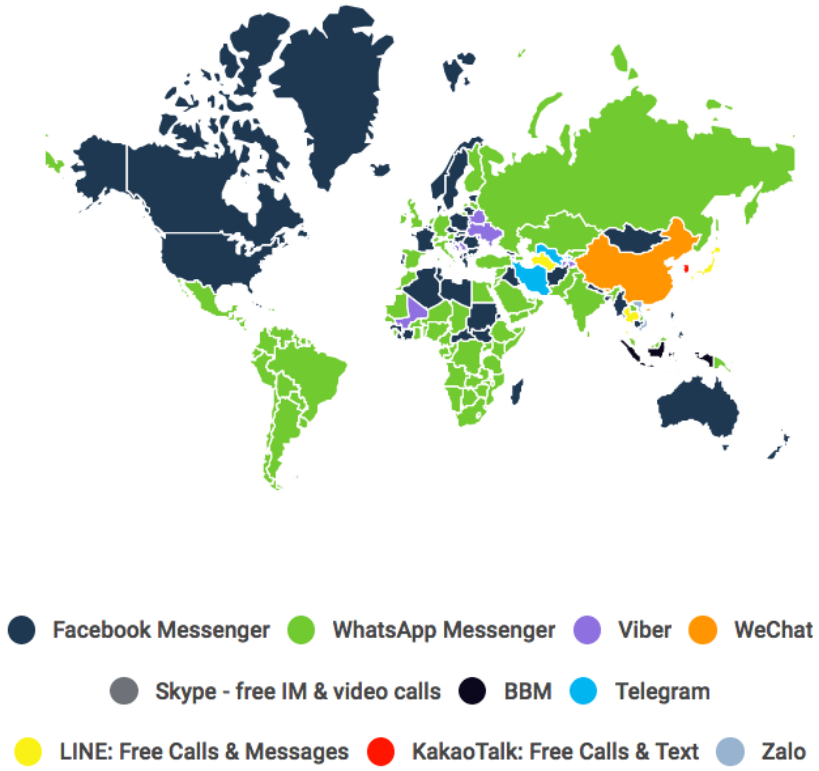

Figure 2.1: Most Popular Messaging App in Every Country[25]

One important factor can be the geographical location of the target audience.
As visible in figure 2.1, Facebook Messenger and WhatsApp are the global leading messengers, and as previously mention the markets in China and Japan is 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.

2.4.1 Cross-platform

As with the creation of other kinds of software it is possible to release the same chatbot software for multiple target platforms. In similar ways to other platforms, the interaction of the software with the platform has to conform to the technical details and protocols of each target, 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 also existing frameworks that allow developers to develop a chatbot once and

release it to multiple platforms at the same time, without any modifications for the individual platforms.
One such framework is API.ai by Google. As of writing they support 16 different integrations, including platforms such as Facebook Messenger, Skye and Slack[26]. However, this platform is more than an adapter to different platforms. It's an 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. The developer can train the bot 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.

Still, there are limits of what can be achieved with a platform 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 response do not match the requirements.
Additionally, while API.ai currently supports 15 different languages, a chatbot is limited to the available languages. If an new language needs to be supported, a lot of work might be necessary switching to a custom solution.
Another 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 extentions might not be available either.
As last point it should be mentioned that, 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 futre. This business model could be charging for the service or, being part of Google, simply collecting data, binding customers and gaining market share. Regardless which monetization strategy is chosen, as a developer one has to be aware that this is a non-controllable, external dependency.

2.5 Classification

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

Chatbot development is still a new market with a lot of changes happening all the time but there are some current trends in what companies are interested in creating. The direction might change soon in the future and there are probably still many undiscovered possibilities.

2.5.1 Categories

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*[27].

Single-feature Chatbots A popular category of bots provide only one single feature. These bots are limited in functionality but simple to use. One example is a Facebook bot called Instant Translator[28]; in the beginning the user selects one language to translate to and all the bot does from there on is to translate all text it receives from the users language to the selected target language.

Proactive Chatbots This category are chatbots that push information to the user instead of answering questions in conversations. Hereby the user doesn't 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 forcast. Another use-case is the Chatbot from the airline KLM[29]; a user can use the service to get updates and information about their flights delivered.

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[30]; when sending a question to Roll the chatbots answers with one name of 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 traditionally require the handling of a lot of bureaucracy. One example is a service called DoNotPay. It is advertised as "the world's first robot lawyer"[31] and the service helps the user with simple legal problems, such as fighting a parking ticket.

Entertainment Chatbots One of the most 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 previous 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 Alex, which have been mentioned earlier, belong to this category.

Optimization Chatbots Most companies are interested in chatbots in this category. The idea is to make an existing product more accessible by creating a chabot for users to connect to the product. By taking advantage of new platforms companies like to reduce friction for customers to use the product. The currently most obvious aspect of the chatbot platforms is the ease for users to access the products. Companies like to optimize the use of their product by making it available via the conversational interfaces of chatbots.

2.5.2 Products

Use cases fitting this category can be found across many different industries. The article "100 Best Bots For Brands & Businesses"[32] 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 big interest in targeting messaging platforms as a new market, the consumer engagement and the development efforts are nothing compared to established markets such as mobile applications or the web. Most of current products only provide limited features and are mostly created to connect customers to existing products.

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

2.6 Promises

As explained previously in 2.2, the interest in chatbots and conversational interfaces increased in recent time, because of the advancements made in the field of artificial intelligence and the popularity of messaging platforms, mobile devices, and personal assistants like Siri.
The conditions are right to think about using the newly available possibilities, however, the question remaining is why we would want chatbots. What can chatbots achieve that existing solutions are not good at, both, from a user's point of view and looking at the interests companies have.

From a user's point of view chatbots can be seen as a new interface to interact with computers. Existing interfaces are not intuitive to humans. Using technology is something humans need to learn first. With every new application one uses and every new website one visits there is a new interfaces 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"[33]. Conversation is a way of communicating humans already know how to use because they use it to interact with other humans. If it is possible to use this communication technology to interact with machines, it should be really intuitive for humans to use it. "The vision for a chatbot: get machines to respond to questions like a human being"[33].

Further, there is a trend in consumer behavior of "outsourcing their "chores", such as driving, shopping, cleaning, food delivery, errands"[34] 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 using a service

is reduced significantly. Managing and coordinating the usage of such services is a task conversational interfaces are particularly suited for because it is a scenario that profits especially from the simplicity and low friction that characterize conversational interfaces.

Users can also profit from technical advantages of chatbots over to native applications and websites.
Native applications need to be downloaded first which includes all resources and not just the ones we require at this moment. Websites are a little slimmer and one only needs to download the resources required to load the current page. 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 the content. Everything else is provided my the platform it is embedded in. Compared to these existing solutions, "a chatbot uses very low bandwidth"[33], which can be an important advantage not only for the perceived responsiveness but especially in places where slow network connections are still common.

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 for the users.

First, "the cost of developing a chatbot is one-third of what is required in developing a mobile app"[33]. This might not be the case for every single product but in general, creating a chatbot is less work than creating a mobile application, because there is no 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"[35]. 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 the flexible nature of using natural language as an interface is that "chatbots are able to gain invaluable data and insights on user behavior"[36], because firstly, users have the freedom to send any kind of information and feedback and secondly, being in the context of a 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 user for further personalization.

Lastly, the most fundamental reasons for a company to be interested in chatbots as a platform are 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"[36]. When the attention of users is shifting away from not only non-digital media but also away from other mobile applications, including traditional social networks, companies need to find a way to reach users at the place they spend most of their time at.

3 Development

To not only understand possible applications of chatbots but also understand what the practical development of a chatbot looks like this chapter guides through the development of an example chatbot.

To start with, an appropriate 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, the appropriate platforms, tooling and solutions can be selected. After all preparations are done the technical implementation will take please. It is followed by any analysis of complications and a comparison to other possible solutions.

3.1 Choosing a Practical Example

Before thinking about implementation details a suitable application for a chatbot needs to be selected.
As illustrated earlier, chatbots can be used to cater a wide variety of applications.
Since this example application should be a suitable 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 2.5, while being, at the same time, not too technical challenging in a problem domain which is not a specific to 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 a technical interesting task to solve regarding image processing, it would not illustrated 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 the section 2.5. Mainly it's a simplification chatbot simplifying the task of language studying, but the chatbot can also be classified as a proactive chatbot because it uses 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 and the main communication medium remains text-based.

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 the implementation of the chatbot features.
In short, a user is able to input new vocabulary; then the system tests the user's knowledge in appropriate learning intervals.

3.2 Existing Solutions

When starting a new projects it is helpful to research for existing solutions that might solve the same problem.
There is already a variety of existing software applications for language studying, which have very different use cases and solve different problems.
One significant separation is between software that includes content and software the user can 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 can be attributed to the ability to sell content. Professionally curating a language course curriculum requires teaching expertise and is a lot of effort, and therefore content 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”[37].
Interestingly, Duolingo recently released a chatbot[38] as part of their iPhone application which enables the user to have a written conversation about certain topics with a chatbot and to thereby learn the appropriate phrases for the given scenario. Although the topic 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 doesn't not provide users a guideline what to study but is instead intended to support users studying their own content.
Most of the software that can be found in this segment is a variation of the attempt to bring traditional flashcards to digital media.
One of the most established software in this segment is Anki[37], which exists for more than ten years already and provides a flexible, but also rather complex, interface to create a various kinds of studying material.
Another more recent competitor is Memrise, where users get a more intuitive interface, which 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 each other, which allows users to reuse what other users created instead.

For the here 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 using the unique features chatbots as a medium provide compared to the here mentioned products.

3.3 Definition of Use Cases

Building on the analysis of existing solutions in 3.2, features for the chatbot need to be specified.
An effective method for gathering crucial features is by finding potential users and creating usage scenarios for their individual needs.

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”[39]

To apply this method, the fundamental problem the application is solving needs to be defined. 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 the learner picked up while studying or in a different situation in every day life.

3.3.1 User Stories

The following are two individuals that might possibly use the chatbot and both of them 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's in the last year for her Bachelor degree in economic. In University she signed up for an evening class in Mandarin. She uses Facebook Messenger every day to talk to her friends and when a friend sent her a link she found the chatbot. 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 Bordaux in France. He studied computer science. A year ago he moved to Berlin where he found a job at a startup. At work everyone all communication is happening 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 and he also tries to remember things he sees in the supermarket. He found the chatbot on a news website for technology products, and since then whenever Pierre learns a new word he gets 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 him 40 minutes each. Now he takes advantage of this time by taking out his phone and reviewing new vocabulary he picked up the previews days.

3.3.2 Functional Requirements

The above defined user stories can be used to extract all necessary functional requirements.

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

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

Last, it is necessary to have a means of deciding what to study next.
A user should not be required to think about what to review or even when it is the right 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, MVP, for this chatbot.
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 similar chatbot products.

3.4 Setup and Requirements

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

3.4.1 Communication

As discussed in 2.3 on page 10, 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 are so basic, there are not many options required and everything can be represented in an unambitious interface.

The precise actions necessary to be defined are 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 the user's guess needs to be evaluated when studying.

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 artificial intelligence based natural language processing techniques, a major source of complexity of chatbot development can be avoided.
While these are useful techniques that enable previous impossible use cases, they are not necessary to explore the fundamentals and paradigms of chatbot development.

3.4.2 Platform

An important question to answer when development a chatbot is which platform to target. As shown in 2.4 on page 11 there are many possible target platforms and some are fundamentally different.

Deciding for a voice-based platform like Alexa, or for SMS or a messenger platform has consequences for all other decisions.

For the case of the example chatbot, while possible, a voice-based interface is rather unsuitable 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, although the prices are pretty low, there are costs for sending text messages. As of writing, Twilio charges $0.0075 for receiving and $0.085 for sending per message when using their Global Short Message Service API in Germany[40].
Supposed the chatbot has 1000 active users that all study 100 phrases daily, the costs would accumulate to $277500 of monthly expenses[2]. These prices are affordable if a company is selling airplane and send tickets directly to users' phones, but in other scenarios another messaging platform is more 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 for interaction available than only plain text. But there are many major existing messaging applications and the choice can be difficult. As we saw earlier, 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 a publicly available API, Facebook Messenger is the biggest platform one can target.
By creating the first version of the example chatbot with English as an interface language, the target markets are mainly North America, Europe and Australia and therefore Facebook Messenger would be a natural fit as target platform.

As mentioned in 2.4.1 on page 14, there are also solutions to create chatbots using a framework which allows to release the chatbot to multiple platforms at the same time, but as previously noted such a framework also has drawbacks.
With the limitations in mind and to keep the example as simple as possible, it will be implemented for a single platform without abstracting the process by using third-party frameworks.

2 $1000 \times 100 \times 30 \times (0.085 + 0.0075) = 277500$

Facebook Messenger is a fitting platform not only due to popularity, but also because it offers mechanisms for interacting with chatbots via interface elements, which will be used in the next chapter to display action buttons in the user interface.

The registration and configuration of a chatbot for Facebook Messenger is not explained in detail here, since the Facebook's own 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 at https://developers.facebook.com/produ
How authentication works is explained in 3.5 on page 30.

Integration

The development of a chatbot for Facebook Messenger is similar to most other platforms.
When creating and configurating a chatbot, the developer registers a webhook URL. "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."[41]
After the setup is done, Facebook will now send an HTTP POST request to the registered URL containing event information for every message a user sends to the chatbot. This way the developer has complete control over how to handle each message on any machine that is reachable via a public URL.

3.4.3 Server

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 to structure the application running on this server.

Programming Language

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

than others though.

In the example case there are not any specific technology requirements.
However, to be able to send notification messages to users when their studies are ready, the system requires a way to schedule timers for the notifications. 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 the pattern of using a separate worker process to handle scheduled jobs, but in this case there would need to be another process for each user just 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[42] or the Node.js JavaScript runtime[43].

The example chatbot is created with the Go programming language[44].
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 further, Go has a robust web server in the standard library which can be exposed to the public Internet without a proxy server, that is common with the previous mentioned programming languages. By making use of this there is less parts to take care of and the example can be a simple and self-contained application.

Data Storage

Some applications do not store any data locally, but only process requests and send out replies.
In the example case data needs to be stored and it should be stored locally on the same machine to keep the application as contained and simple as possible.

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 the user does while studying and to decide when to study next,

the time of the last study needs to be tracked too.

To send notification 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 not seen the last notification yet, no new one has to be sent either.

All of the information required to be stored is always focused on a user. Users can never share information with each other. There are no complex relations between data.
Without relational data there is not much use of features that relational databases such as MySQL or PostgreSQL provide; a simple key-value store can be sufficient for storing this data instead.
The example uses an embedded key-value store called BoltDB[45].
No separate process is needed for the database and data is saved on disk in a single file.
These technical decisions are the base for the following description of the chatbot implementation.

3.5 Feature Implementation

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

Before looking at the implemenation, the usage of the chatbot is shown from a user's point of view.
The following is a presentation of how this chatbot is used.

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

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 administror, the bot can be found by using the

search as shown in figure 3.1.

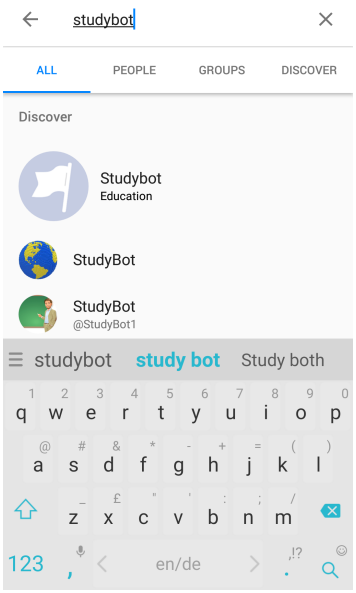

Figure 3.1: Search

After navigating to the chatbot, a description of the bot becomes visible. This can be seen in figure 3.2.

This view 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 view.

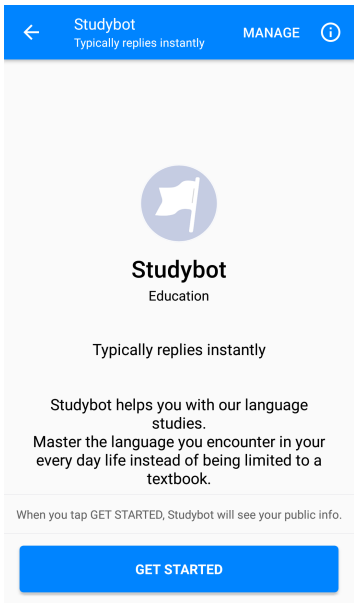

Figure 3.2: Get Started Screen

When pressing the **Get Started** button, a message is sent to the chatbot, and as indicated by the *dots* visible in figure 3.3 in 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 text.

For chatbots the *dots* indicated that the chatbot received the message and is crafting a reply.

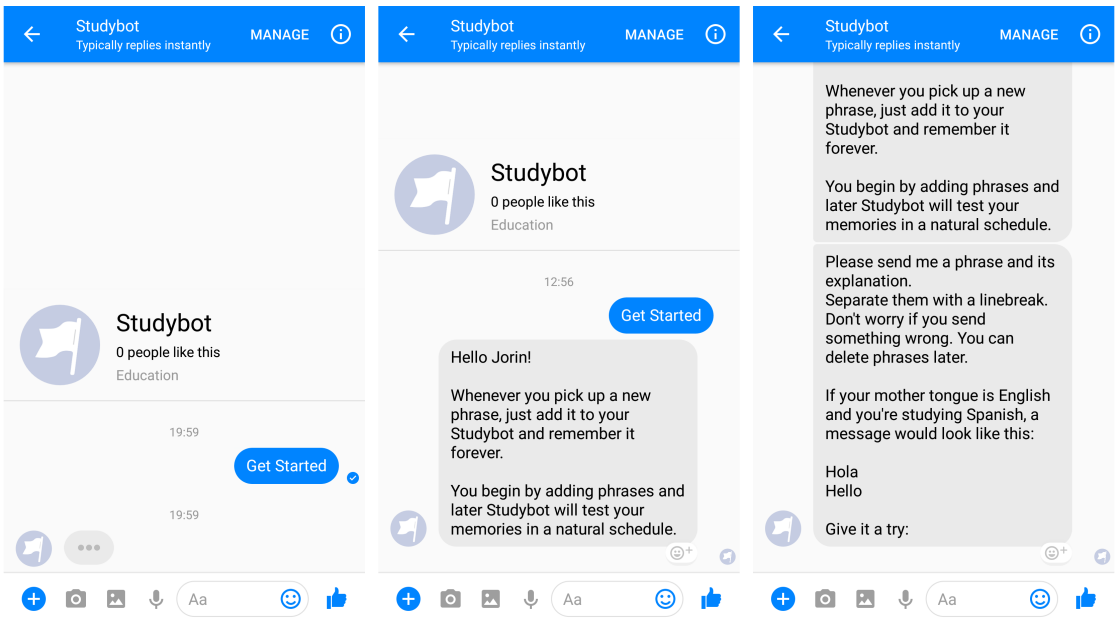

Figure 3.3: Messages introducing Studybot

The image in the middle of figure 3.3 shows the first message sent to users; users are greeted using their first name to create a more personal feeling atmosphere. The greeting is followed by a two sentence long explanation of what this chatbot is doing.
When working with text as a medium it is especially critical to focus only on the most important information; with graphical elements and images users can perceive an impression with a single glace, but to perceive the meaning of text the user needs to read word by word. Since 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 3.3 that a second message is sent to the user. The second message is delivered 5 seconds later that the first one to not overwhelm the users with too much information at once by putting a *wall of text* on their screen.
This message contains instruction for the first step of using the application.
The instructions are additionally illustrated by providing an example for a message in the expected format.

At this point the user needs to interact with the chatbot.
Figure 3.4 shows how a user adds phrases to Studybot; when typing a phrase, users first

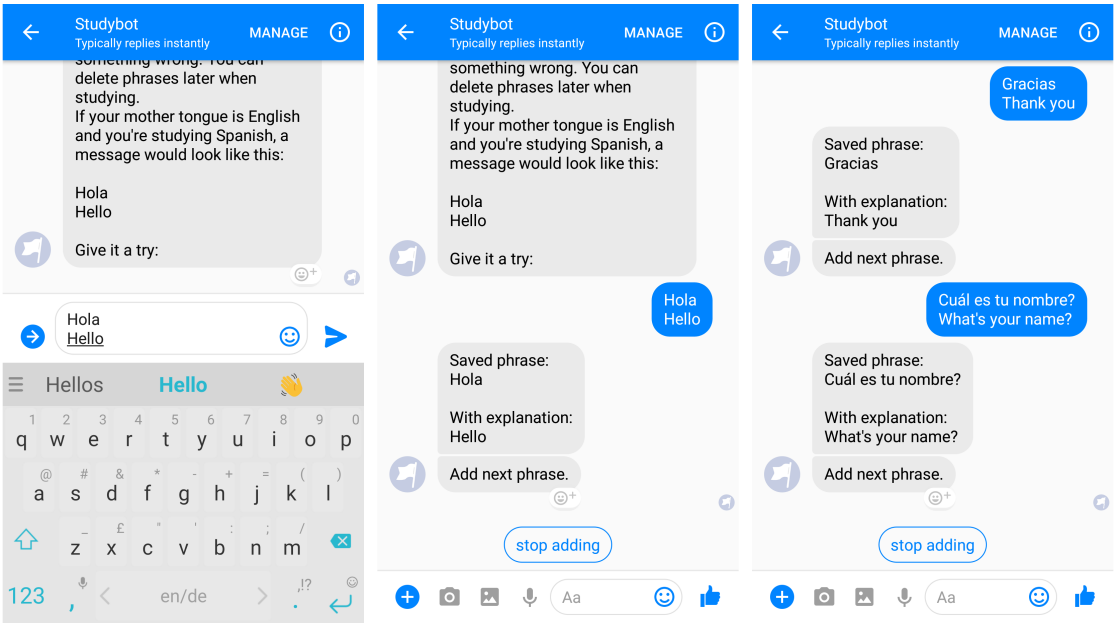

Figure 3.4: Adding three phrases

types the phrase in a foreign language they like to study, then the *return* or *enter* key on the keyboard has to be used to insert a *line-break*, 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 a message should most of the time end with a prompt for a certain input or action.

The image in the middle of figure 3.4 shows that this is the first point of the conversation, that in addition to being able to type a reply, a button is displayed at the bottom of the messages, which the user can use as an alternative way for interacting with the chatbot.

As shown in figure 3.4, three phrases have been added to Studybot before the user decides to press the button labeled as **stop adding**.
Figure 3.5 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**

respectively and each button label also contains an emoji[3] used as icon for the action to make it easier identifiable for the user. When relying on mainly on text for communication and with limited usage of graphical elements, emojis can be a helpful substitution of traditional icons to provide more visual guidance for users.

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 address messages in the correct context.
In figure 3.5 the user has left the mode for adding words and entered the *menu mode* from where other modes can be accessed.

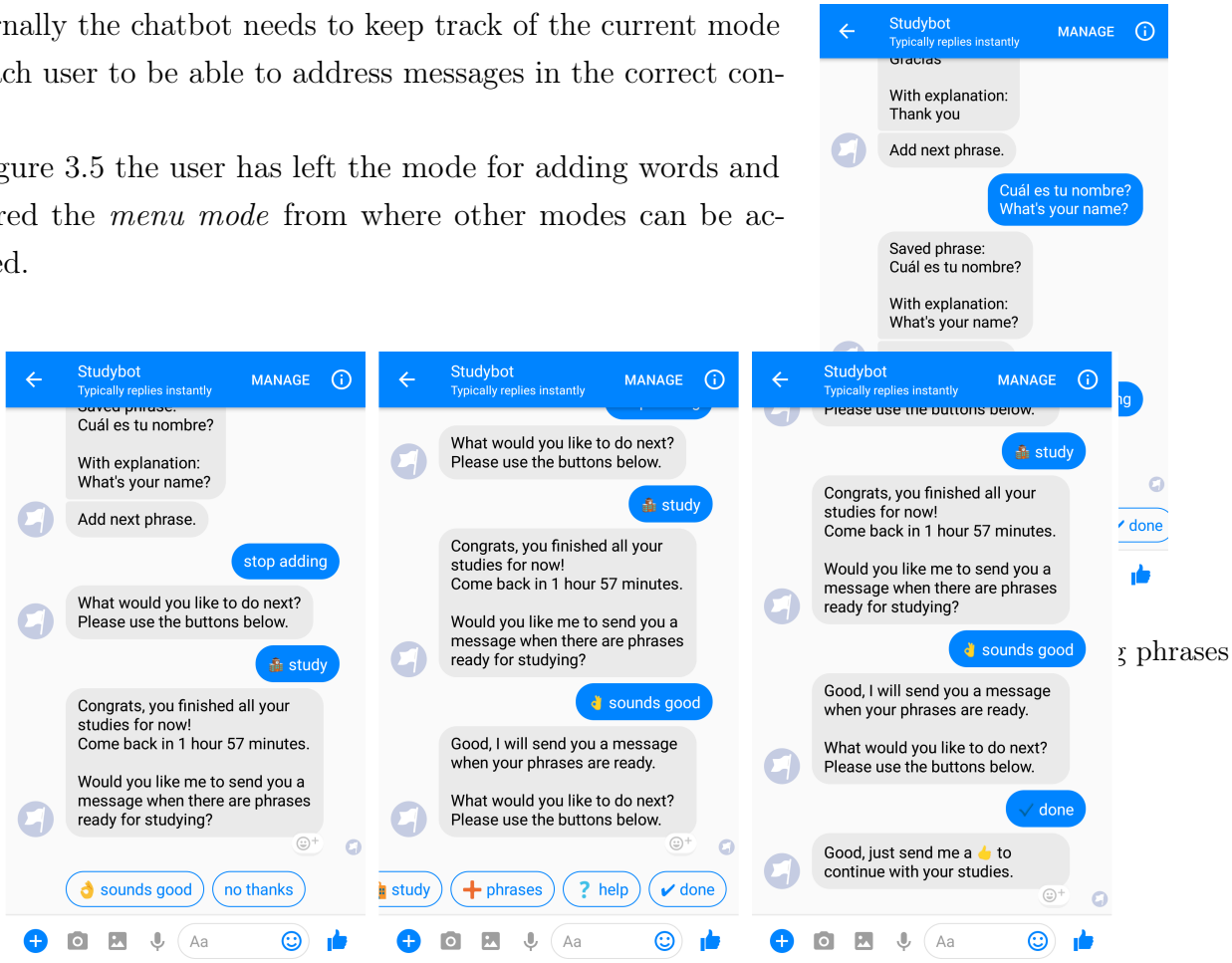

Figure 3.6: Attempt to study and activation of notifications

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 3.6 shows, the added

3 "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."[46]

phrases cannot be studied yet.

Studybot uses a concept known as *spaced repetition system*, or short *SRS*, which is described as "a learning technique that incorporates increasing intervals of time between subsequent review of previously learned material in order to exploit the psychological spacing effect."[47]
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.

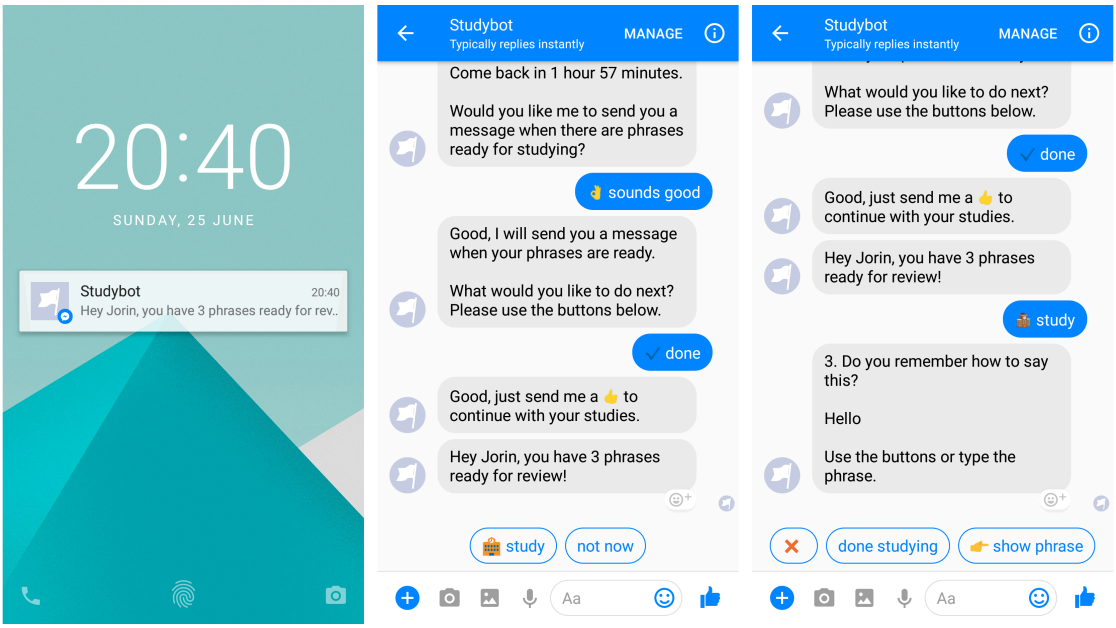

Figure 3.7: Notification message and beginning of study

Instead of studying the previously added phrases directly, the user is offered to receive a notification message once phrases are ready for review.

When sending notifications to users, it is important to not send more messages than necessary, and depending on the messenger platform, there are different policies in place on how many messages are permitted.
The platform policy of Facebook Messenger clearly states 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"[48], which means there needs to be a way for users to disable the sending of notifications; and 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."[48]
How to disable notifications 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 46.

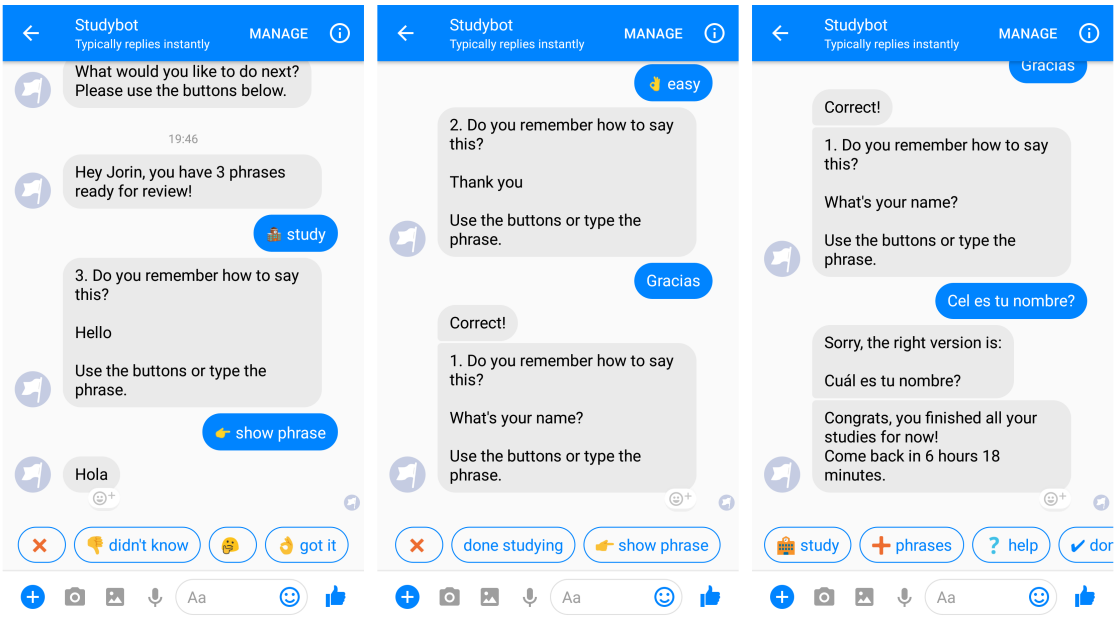

Figure 3.8: Study using buttons or by typing

In figure 3.7 a notification in from the Messenger application on the mobile phone is shown. It is triggered when studies are ready for review.
When opening the conversation with the chatbot, the user is now prompted to study.
As defined as a requirement in 3.3.2 on page 25, 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 3.7, allows users to delete a phrase.
Since there is no possibility in Facebook Messenger to display a long list of interactive elements to the user, creating separate functionality for editing and deleting phrases is difficult to achieve, and while studying is at this point the only possibility to refer to a phrase.

The figures 3.7 and 3.8 show a subtle decision to display the buttons users are most likely to interact with, on the right side; in other places it is more intuitive to show the most important

information on the left, since English language is written from left to right and we scan text accordingly; however for certain interactions, like studying in this case, need to be performed so frequently, a user will remember the location of the buttons quickly and doesn't need to scan process the information every single time.
In this case it is helpful to have the most used action on the right side, since the majority of humans is right-handed, therefore can reach the button with the thumb with less effort when using a mobile phone single-handed.

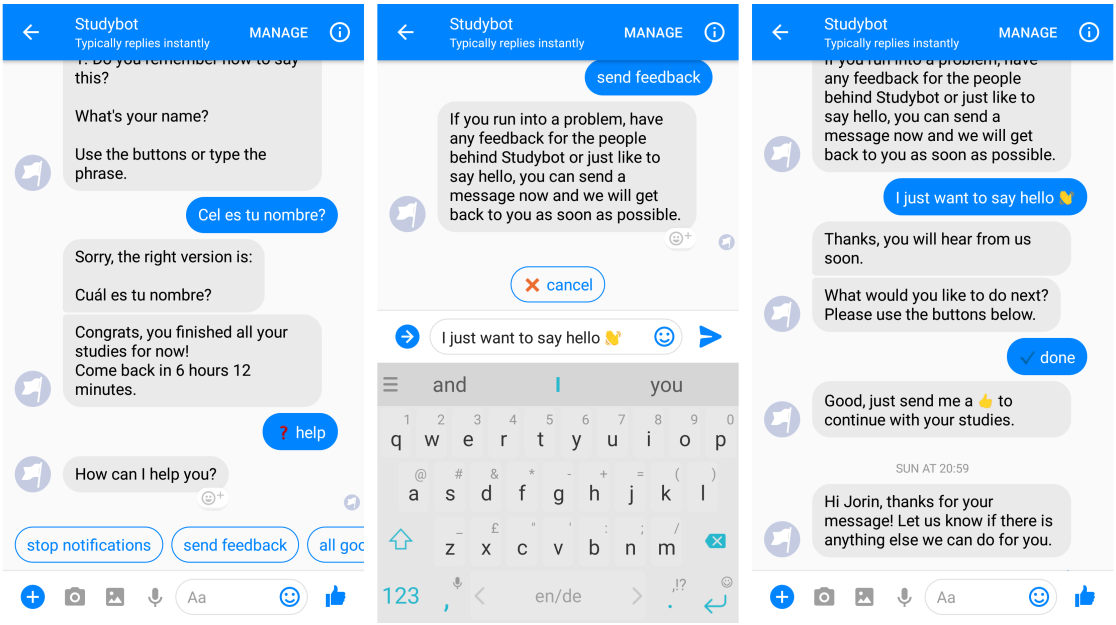

Figure 3.9: Send feedback and receive a reply

A feature, that is outside of the main flow of using the chatbot, is the sending of feedback to the developers of the chatbot.
As shown in figure 3.9, 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 replies are sent automated by the chatbot, but there should be a way users can talk to the developers or administrators that provide the chatbot.
In the particular case of this chatbot the feature is automated by sending the messages users send as *feedback* to a Slack channel[49]. Additionally administrators and developers can directly reply to user messages inside Slack and the messages is forwarded to the correct Facebook user within the chatbot.
Figure 3.9 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.

With this the primary interactions for the example chatbot are covered.
Many of the implemented interactions can be transfered and reused when creating other kinds of chatbot; addressing the user by name, sending text in small chunks with a time of delay in between, prompting the user for specific input with every message, keeping track of the user's context, asking for permission before sending notifications, consciously ordering buttons and allowing 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 the task done, which is expressed well in the following quote from the newspaper the Guardian[50] about the lessons they learned from 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."

3.6 Server Architecture

The following is an overview about how the example chatbot 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 similar patterns in their own development.

There are numerous resources available today that demonstrate the development of chatbots. However a majority of the existing material relies on specific services, frameworks or libraries for the implementation.
The implementation of the example chatbot demonstrates all basics necessary for the creation

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

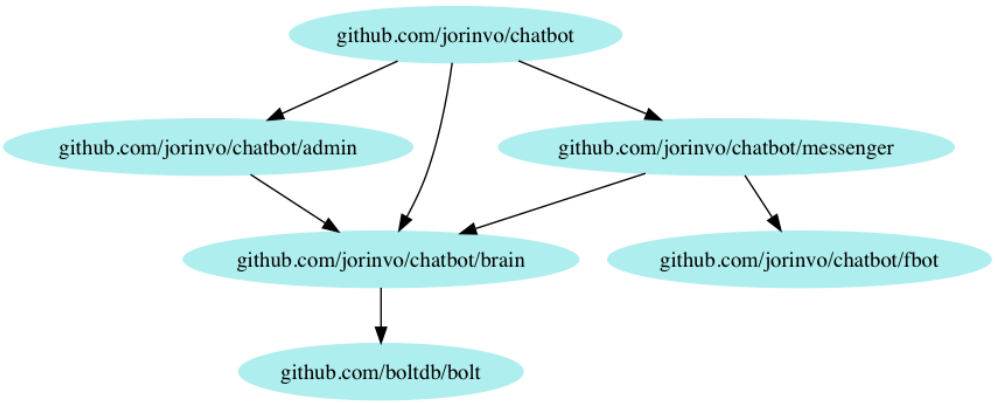

Figure 3.10: Package dependency graph[4]

of a chatbot without relying on external tooling; the code-base has no external dependencies, with the exception of a package for the database.

The Graph in figure 3.10 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 fulfill their HTTP requests 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 3.5 on page 37 mentioned feature.

Package **messenger** is responsible for processing all events received from the Facebook Messenger platform.

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

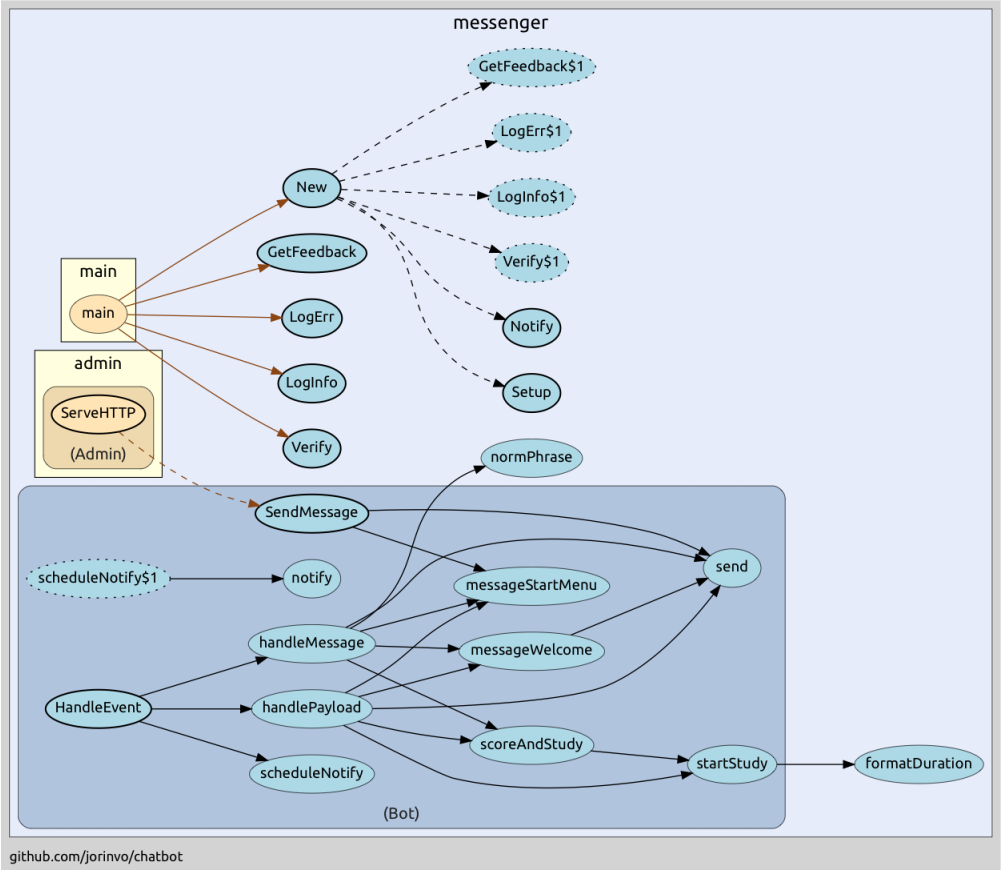

Figure 3.11: Call Graph of the messenger package[5]

Most of the code-base and its architecture are structure the same way most other servers are organized, the logic that is more specific to the development of a chatbot is mainly contained in the package **messenger**.
Figure 3.11 illustrates the behavior of this package, but it should be noted that for brevity calls to the packages **brain** and **fbot** are not included in the graphic.
The upper half of the graphic in 3.11 shows function calls, which 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 sending properly generated responses back to users.
The function *HandleEvent*, which can be seen in figure 3.11, is called by package **fbot** with every event the webhook receives, and this is the main entry-point into the custom chatbot logic. Depending on the type of the event, *HandleEvent* tracks whenever a user read a

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

message, or sends a message back to the user, whereby it has to be differentiate between handling raw text messages or handling the use of a predefined button and its action payload.

In addition to directly responding to users, *HandleEvent* is also the source for scheduling the sending of notifications. With every action of a user, the next time they receive a notification message needs to be rescheduled, which happens in the call to the function *scheduleNotify*. The function in 3.11 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 the user.

Further, it is visible in 3.11, that package **admin** can call a functioned named *SendMessage* which belongs to the *Bot* type. This is used to send replies, that administrators created inside Slack, back to users, since package **messenger** is the only package responsible for communicating with the Facebook Messenger platform.

Limiting 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 47, additionally the Go documentation for all exported types and functions can be found on page 49, which explain the actual implementaion in more detail.

3.7 Limitations

In the previous section different properties of using a chatbot as medium and solutions for certain scenarios based on those properties have been mentioned.

There are further properties that affect the design of chatbot based systems, and which can in some cases even frame certain systems as unfeasible.

One fundamental property of the implementation of chatbots, is that they are not software running on the devices of users. They mimic the nature of other chat situations where the counterpart a user is communicating is not on the same device as the user is using.
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 the network connections are part of the Internet and such a system can be described as a

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

First and foremost this implies that all usage scenarios which are obliged to work without any network connection cannot make use of chatbots. Exemplary areas of applications affected by this requirement are applications that forbid network connections due high security standards and also products specializing in usability in rural, remote location with unstable network connections.

Another implication is the unavoidable latency of networked applications.
When information needs to be transfered from one device to another, it needs to be transported through a larger physical.
For the example chatbot, which is 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 from Facebook.
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, these 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 suited for any time-critical applications, and it can not be guaranteed that a user receives within a 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 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 limits interactivity. While this can be a very simple and efficient way of communicating

for 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 media are for example photo and video editing applications or also 3D gaming.

Due to the limitations the medium chatbot has, it can be anticipated, that, while they can address many needs for humans to communicate with computers, they are not universally applicable and they are not able to replace all previous 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.

4 Conclusion and Outlook

- "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." http://whoo.ps/2015/02/23/futures-of-text
- chat is the new command line: linear flow resembles command lines developers use to interact with computers, but a more human understandable approach. computing started with command lines, then GUI was invented and we forgot about text, now we have to combine them. but command lines are especially powerful because we can combine commands, this is something that could happen in the future with open platforms instead of separate chatbots.
- text should be enhanced with UI: conversational interfaces engage with a story as main flow but doesn't have to be natural language and AI. Haven't touched the topic of AI-powered chatbots here, because fundamentals of the medium remain the same without. AI can be used to make it more flexible and to solve more complex scenarios. Actually conversation as only form of interaction is hard; we struggle so often to talk to other humans, with computers we have the advantage of not being limited to only conv. When talking to people we often draw things on a blackboard to clarify it, the same way we also should assist conv with computers by augmenting it with some helpful elements.
- Text is not the best solution for everything. It's only good if you need flexibility. Mostly users are lazy. Chatbots exist for a long time but just now there are ways to make them more usable by helping users to be lazy. "If something can be tapped/clicked instead of typing, they prefer that." https://chatbotslife.com/dear-bot-developers-dont-get-over-hyped-374572412fda "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,"[33]
""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."" https://chatbotsmagazine.com/does-a-bot-need-natural-language-processing-c2f76ab7ef11
- it's story telling: as a developer, there is no need to worry about interface design, still need to worry about interaction design though; it is a lot like telling a story to guide user through the program, just way less visual
- usage scenarios: works well for the example scenario, section about products show other good candidates: "Another 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." https://chatbotsmagazine.com/does-a-bot-need-natural-language-processing-c2f76ab7ef11 strong for: Sales, News, Travel promising: Healthcare, Banking https://blog.botlist.co/these-2-industries-are-the-next-big-things-for-chatbots/
- Unify bots in something like Siri, Google Search or https://luka.ai/
- "messaging as an input method" "Imagine if services could even respond directly to my input" http://whoo.ps/2015/02/23/futures-of-text https://core.telegram.org/botsinline-mode
- don't call it chatbot ""Chatbot" sets bad user expectations" "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." ""Bot" is a short, memorable, and neutral." ""It's not what you say, it's what the other person hears."" "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." http://botnerds.com/chatbots/
- Moving all our computing into the cloud and depending on closed chat platforms for everything at least support multiple so users have a choice

A Figures

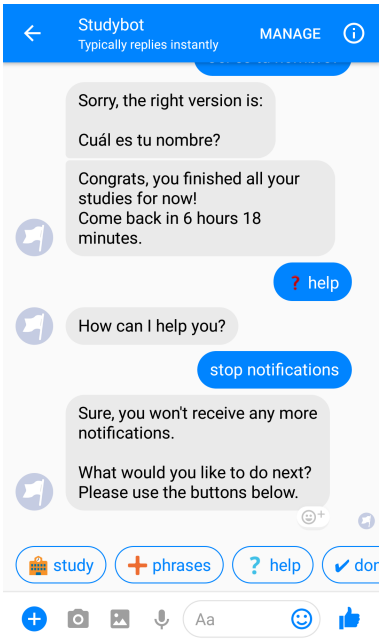

Figure A.1: Disable notifications

A.1 Call Graphs

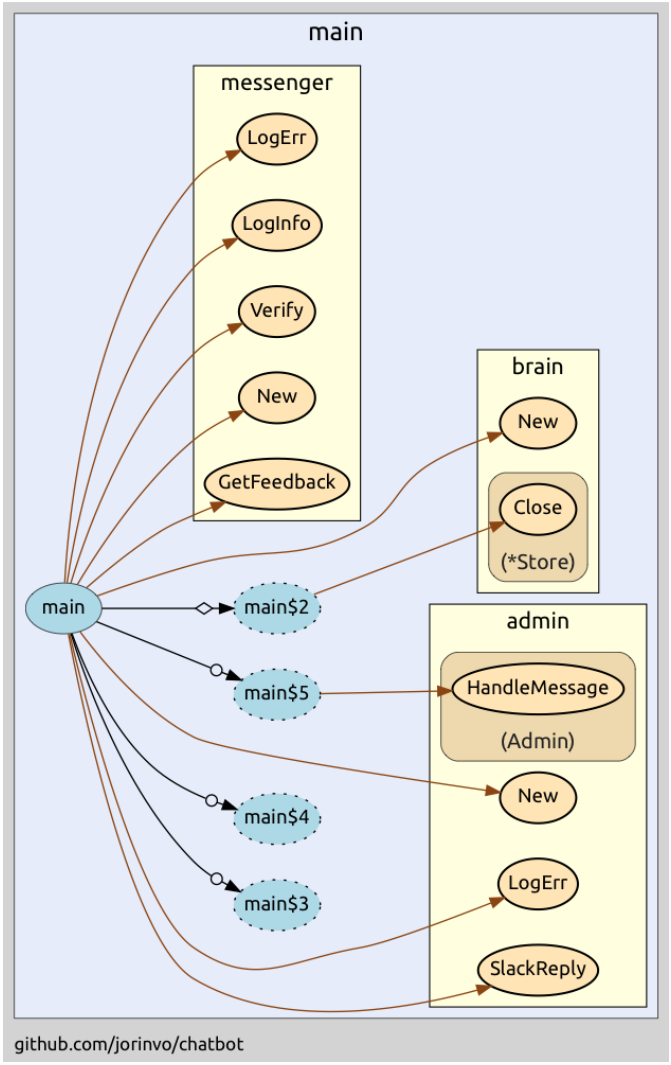

Figure A.2: Call Graph of package main

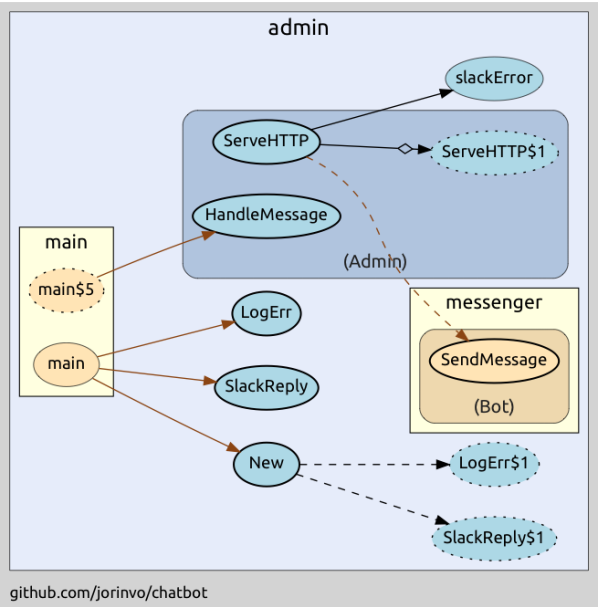

Figure A.3: Call Graph of package admin

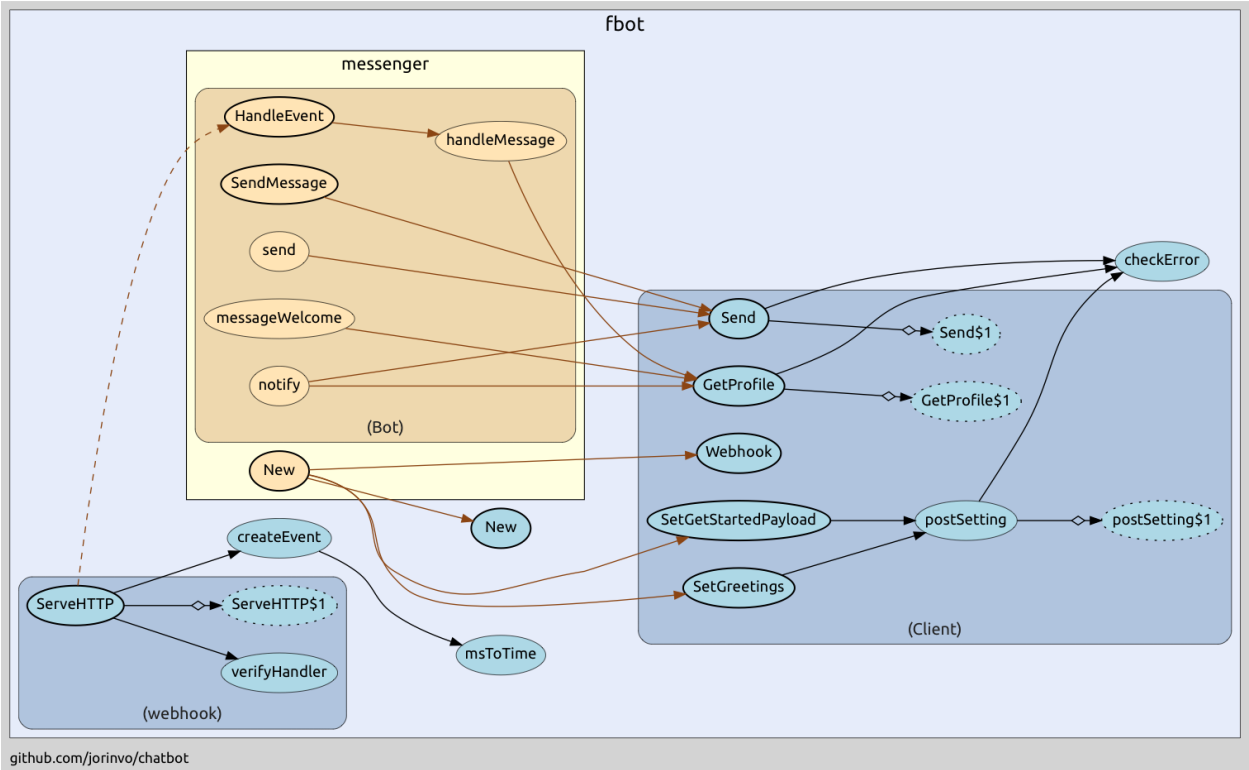

Figure A.4: Call Graph of package fbot

A.2 Documentation

Listing A.1: Command line interface usage information

```
Slangebrain Messenger bot

Usage: chatbot [flags]

Slangbrain 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.

Slangbrain 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 A.2: Directory listing

```
/Users/jorin/go/src/github.com/jorinvo/chatbot
           admin
                   admin.go
           brain
                   brain.go
                   mode.go
                   phrase.go
                   store.go
                   study.go
                   subscription.go
                   util.go
           call-graph-admin.png
           call-graph-brain.png
           call-graph-fbot.png
           call-graph-main.png
           call-graph-messenger.png
           fbot
                   fbot.go
                   profile.go
                   response.go
                   settings.go
                   webhook.go
           main.go
           messenger
               bot.go
               buttons.go
               messages.go
               messenger.go
               notify.go
               payloads.go

4 directories, 25 files
```

Listing A.3: Go documentation for package messenger

```
1  package messenger
2      import "github.com/jorinvo/chatbot/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 A.4: Go documentation for package admin

```
1  package admin
2      import "github.com/jorinvo/chatbot/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 A.5: Go documentation for package fbot

```
1  package fbot
2      import "github.com/jorinvo/chatbot/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 A.6: Go documentation for package brain

```
package brain
    import "github.com/jorinvo/chatbot/brain"

    Package brain handles all business logic of Slangbrain. It handles data
    storage, retrieving and updating. It's independent from the used bot
    platform and user interaction.

TYPES

type Mode int
    Mode is the state of a chat. We need to keep track of the state each
    chat is in.

const (
    // ModeMenu shows the main menu.
    ModeMenu Mode = iota
    // ModeAdd lets the user add new phrases.
    ModeAdd
    // ModeStudy goes to phrases ready to study.
    ModeStudy
    // ModeGetStarted sends an introduction to the user.
    ModeGetStarted
    // ModeFeedback allows the user to send a message that is ready by a human.
    ModeFeedback
)

type Phrase struct {
    Phrase      string
    Explanation string
    Score       int
}
    Phrase describes a phrase the user saved.

type Store struct {
    // contains filtered or unexported fields
}
    Store provides functions to interact with the underlying database.

func New(dbFile string) (Store, error)
    New returns a new Store with a database already setup.

func (store Store) AddPhrase(chatID int64, phrase, explanation string) error
    AddPhrase stores a new phrase.

func (store Store) BackupTo(w http.ResponseWriter)
    BackupTo streams backup as an HTTP response.

func (store Store) Close() error
    Close the underlying database connection.

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] *Turing Test*. URL: http://www.turing.org.uk/scrapbook/test.html (visited on 05/03/2017).

[7] *Eliza Test*. URL: http://www.masswerk.at/elizabot/eliza_test.html (visited on 05/03/2017).

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

[9] 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).

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

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

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

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

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

[15] *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).

[16] *Machine learning - Explore - Google Trends*. URL: https://trends.google.com/trends/explore?date=all%5C&q=%5C%2Fm%5C%2F01hyh_ (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] *Messenger Bots for Business & Developers*. URL: https://messenger.fb.com/ (visited on 05/03/2017).

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

[24] 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).

[25] 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).

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

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

[28] 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).

[29] *KLM on Messenger*. URL: https://messenger.klm.com/ (visited on 05/04/2017).

[30] 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).

[31] 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).

[32] 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).

[33] 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).

[34] 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).

[35] 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).

[36] 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).

[37] 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).

[38] 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).

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

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

[41] *SMS Pricing for Text Messaging*. URL: https://webhooks.pbworks.com/w/page/13385124/FrontPage (visited on 06/24/2017).

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

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

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

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

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

[47] *Spaced repetition - Wikipedia*. URL: https://en.wikipedia.org/wiki/Spaced_repetition (visited on 06/27/2017).

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

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

[50] *What The Guardian has learned from chatbots*. 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).

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