Bachelor Thesis

BSc in Computer Engineering

On the use of conversational agents to measure caregiver burden

Conversational Agent that measures caregiver burden through psychological test questions introduced in a normal conversation environment.

Author
Eugenia Castilla Fragoso

Tutor
Oresti Baños Legrán

School of Technology and Telecommunications Engineering of the University of Granada

—

Granada, September of 2020

ON THE USE OF CONVERSATIONAL AGENTS TO MEASURE CAREGIVER BURDEN: Conversational Agent that measures caregiver burden through psychological test questions introduced in a normal conversation environment.

Eugenia Castilla Fragoso

Keywords: chatbot,conversational agent,the Zarit Test, natural language, Dialogflow, Telegram and Firebase.

Abstract

As mental health disorders keep on affecting more people world wide, more families see themselves as caregivers. Most of the times, these family members have not received a proper training and find themselves struggling to make ends meet, experience loneliness, abandonment, and often develop signs of depression. All the symptoms described can be summarized in what is called the caregiver burden syndrome. The attention given to these caregivers is scarce, but the technology of nowadays offers a new opportunity to provide help and attention to caregivers. This project aims at developing a conversational agent able to measure caregiver burden through psychological test questions asked as an easy going conversation takes place. In terms of tools, to develop this project Dialogflow has been used along with Firebase and Telegram. The psychological test used has been the Zarit Test Burden Interview, and to introduce the questions in a normal conversation the interview has been divided into groups of similar contexts. Preliminary results found the system useful and effective. The evaluation also states the chatbot as an opening door to more developments of this kind, proving the possibilities of conversational agents in mental healthcare.

I, **Eugenia Castilla Fragoso**, student of the degree in Computer Engineering of the School of Technology and Telecommunications Engineering of the University of Granada, declare that the present End of Degree project is original, having not used sources without being duly cited. If I do not comply with this commitment, I am aware that, in accordance with the Evaluation and Qualification Regulations of the students of the University of Granada of May 20, 2013, this will automatically entail the numerical rating of zero [...] regardless of the rest of the qualifications that the student would have obtained. This consequence should be without prejudice to the disciplinary responsibilities that students who may commit plagiarism may incur. In addition, this same work is published under the Creative Commons Attribution-ShareAlike 4.0 CC license, giving permission to copy and redistribute it in any medium or format, also to adapt it in any way you want, but all this as long as the authorship is recognized and distributed with the same license as the original work.

Sgd: Eugenia Castilla Fragoso

Granada on September 08, 2020.

D. **Oresti Baños Legrán, professor in the Department of Computer Architecture and Computer Technology at the University of Granada.**

Informs:

That the present work, entitled ***ON THE USE OF CONVERSATIONAL AGENTS TO MEASURE CAREGIVER BURDEN,Conversational Agent that measures caregiver burden through psychological test questions introduced in a normal conversation environment***, has been carried out under his supervision by **Eugenia Castilla Fragoso**, and authorizes the defense of such work before the appropriate court.

Issues and signs this report in Granada on September 08, 2020.

The Tutor:

Oresti Baños Legrán

Acknowledgements

I would like to thank my parents, who have supported me and listened to all my "computer problems" even if they did not understand a thing. Thank you to my dearest friends, who in times of doubt and weakness gave me strength to continue, and thank you to my tutor for all his patience and support.

This project would have never happened without the inspiration of my grandparents, Francisco and Pili. He fights Alzheimer’s every day and she loves and cares for him every second. Their love and resilience has deeply influenced this project, I can only hope that someday people like them get the true help they deserve.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

According to the World Health Organization [1], one out of four people in the world will be affected by mental or neurological disorders at some point in their lives, and 450 million people currently suffer from such conditions. As reported by the *Alzheimer's association* [23] in their 2020 Alzheimer's Disease Facts and Figures Report, more than 5 million Americans are living with Alzheimer's. By 2050, this number is expected to rise to nearly 14 million. One in three seniors die with Alzheimer's or another dementia. These diseases kill more than breast cancer and prostate cancer combined. There is no question that finding a cure to these disorders is a world wide mission, but as the task is taking place, the carers of patients that suffer these illnesses are often unattended.

From a young age we hear about these disorders but unless we unfortunately get the chance to work with a patient ourselves, information is very limited. Furthermore, not only are mental health disorders one of the fastest growing conditions, but they are also the ones that receive most of the caring from family members at some point, family members that are not properly trained. According to the 2020 report of Alzheimer's disease facts and figures of the *Alzheimer's association* [23] more than 16 million Americans provide unpaid care for people with Alzheimer's or other dementias, these caregivers provide an estimated 16.6 billion hours of care valued at nearly 244 billion dollars. More shockingly, 83% of the help provided to older adults in the U.S.A comes from family members and nearly half of all caregivers who provide help to older adults do so for someone living with Alzheimer's or another dementia.

As expected, family members that have to care for their loved ones often suffer themselves symptoms of depression, anxiety, etc produced not only because of the pain of having to see a loved one go, but also because of the burden and expectations that come with the job. Fortunately, caregivers are getting more attention, as it has been proven that depressive symptoms among patients are associated with the mental state of caregivers [31].

Over the years, many tools have been developed to assess the level of burden of the caregivers, but these have many complications. For one, these tests are often carried out sporadically if lucky, and they consist of very personal yet efficient questions that become hard to answer, especially if they have to be faced on a short period of time. One of the best known test is the Zarit Test, an interview that consists of 22 questions, reporting on topics like economic means, personal experiences and expectations. The test offers very personal questions that are often hard to face, but its proven effectiveness in evaluating the level of burden in caregivers makes it an adequate tool to use in this project. The caregiver needs to evaluate each statement from a range of 0 to 4, 0 being never and 4 nearly always.

1.1 Motivation

In today's world, technology gives us the chance to improve life. In particular, with the development of chat bots, also known as digital conversational agents, we have a new way of keeping in touch with people. So why not take advantage of technology to stay closer to caregivers? It would be wise to say that now is the time to unite once more the old with the new, and to use modern technology like conversational agents to take care of those who tend to others. The use of such technologies allows us to embed these type of questionnaires in a ubiquitous, continuous, personalised and non-invasive manner as part of regular conversations which are not only aimed at measuring the caregiver burden state but also alleviate it to the possible extent.

1.2 Objectives

The goal of this work will be to introduce a digital conversational agent that is able to provide with a personal experience for caregivers while

taking the test-like questionnaires, with the goal of measuring the quality of life and burden of caregivers.

To achieve this goal, the following supporting objectives are identified:

- Design a digital conversational agent that introduces questionnaires.
- To implement a medical consultant system service by using chatbot technology that studies the level of burden and quality of life of caregivers of patients with dementia.
- Evaluate the quality of life and burden of caregivers.

1.3 Planning and Budget

A Gantt diagram 1.1 was created for this project. The diagram was edited during the proceedings to adapt it to timelines. The tool `Team Gantt` [12] was used to create this diagram.

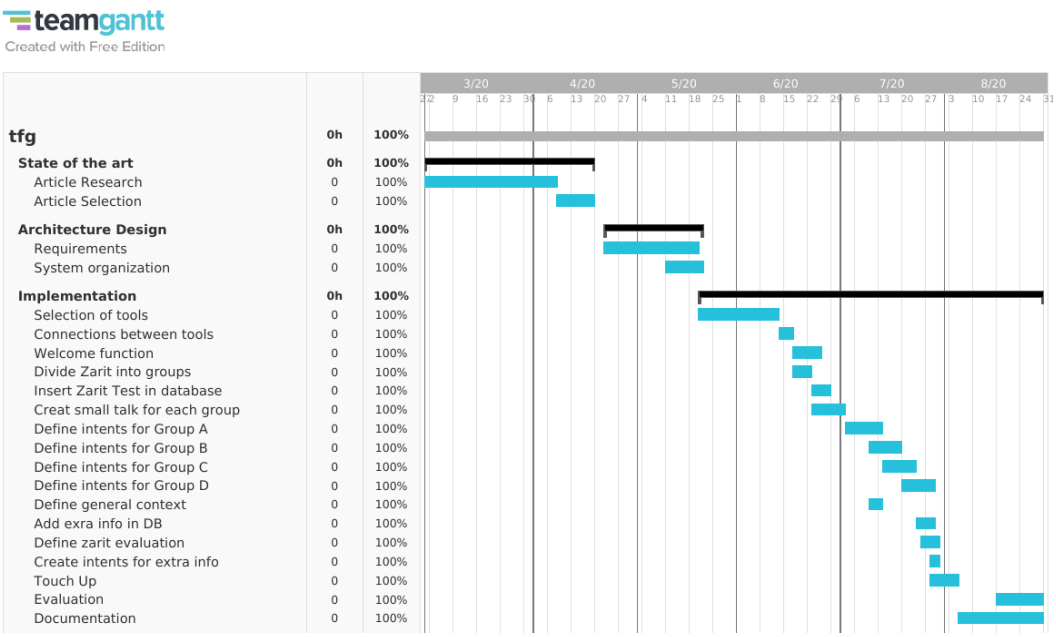


Figure 1.1: Use case B: The chatbot initiates the interaction.

The estimated price of the development of this project was established at 13.340,25€. Table 1.1 showcases the budget breakdown for

the project.

Description	Hours	Price per hour	Total
App Development	550 aprox.	20 €	11.000 €
Tools			25€
Taxes | 21%			2.315,25€
Total			**13.340,25€**

Table 1.1: Budget of project.

1.4 Structure of the thesis

The documentation of this projects is set as follows:

- **Chapter 1**(Introduction): introductory section to the project, stating the motivation, objectives and the planning and budget for this project..
- **Chapter 2**(State of the art): this section includes different researches and articles relevant to the project. Publications that have served as inspiration and proof are included in this section.
- **Chapter 3**(Design): this chapter describes the design of the project. It is formed by the requirements, the architecture.
- **Chapter 4**(Implementation): all the information regarding the implementation of this project is held in this chapter, this includes the specification of the tools used.
- **Chapter 5**(Results): the results of the preliminary study of effectiveness of this project are included in this chapter, as well as the personal reviews of the users and their demographics.
- **Chapter 6**(Conclusions and future work): this chapter includes a section on conclusions according to the results received, a review of the requirements achieved in this development, as well as a section on future development opportunities.

Chapter 2

State of the art

2.1 The Zarit Test and The Caregiver Burden Syndrome

Mental health disorders are not new to our society, and although the Zarit Burden Interview, also commonly known as the Zarit Test, has been around since 1980, it is not very known outside of the medical community. It is important to know that in 1963, Grad and Sainsbury published the first conception of the caregiver burden syndrome, and that it was not until 1980 when the Zarit Test was published. This test provides a comprehensive assessment of the feeling of burden of a caregiver. It is very well considered in the medical community, thus it has been translated into many languages, though for the purpose of this paper, we will be using the English official translation. There is no sense in continuing to talk about the Zarit Test without first assessing the actual meaning of the caregiver burden syndrome. The caregiver burden syndrome is defined by George and Gwyther [18] as "the physical, psychological or emotional, social and financial problems that can be experienced by family members caring for impaired elderly adults". It is fair to say that caregiver burden syndrome can appear on any person caring for an elderly relative, no matter the sickness of that relative. This syndrome can appear even in those caregivers of elderly people that have no major disease to manage. There are some studies however, that indicate that the level of burden increases in a caregiver when diseases like Alzheimer's come in play. Gonzalez et al. [19] conducted an study to asses the stress and psychological morbidity of Alzheimer patient caregivers. To

do so, they studied the cases of fifty-eight caregivers of Alzheimer's patients and 32 caregivers of non-demented chronically ill patients. Theirs results established that the stress experienced by caregivers of AD patients was higher than in the other group. So was the psychological morbidity. It has been also proved that in some cases, the negative effects of caring for an AD patient may keep on being present in the caregiver for up to 12 months after the patient has deceased [22].

Keeping all this data in mind, the existence of the Zarit Test Interview is more than justified. Seng et al. [49], in 2010, developed a study to determine the validity and reliability of the Zarit Test in assessing care-giving burden in specifically caregivers that were family members of the ill ones, not professional carers or friends. They concluded that the Zarit Burden Interview was a valid and reliable instrument. Their article also mentions that the study was conducted both face to face and telematically, which one would have thought that would make the test harder for those who did not have immediate support if they did not understand a question, but this was not the case; leading to the conclusion that the English version is clear enough to be understood without much issue. Once we have the syndrome and the tools to identify it, some scientist have investigated different methods of coping with it; as is the case of Cooper and peers [11], who concluded that the best practices to cope with the burden were emotion-focused strategies, whereas those who used problem-focused strategies were more anxious 1 year later. The emotion-focused mechanisms included exercises for acceptance, humour, positive reframing. Problem focused strategies included active coping practices or planning. Religion is also a very popular and effective coping mechanism among caregivers according to Stolley et al. [43]. Information about institutionalization will not be included, as some studies suggest that there is no relation between the relief of the caregiver and the institutionalization of the patient (Pratt et al. [38]).

2.2 Conversational agents

Conversational agents have been around for a while, thus, the technology behind them has developed along the years. There are several chatbots available, but all of conversational agents can be identified within one of these two groups: rule-based chatbots or artificial intelligence chatbots. Rule-based chatbots use rules to determine its behaviour. Even if the first chatbots were created using this line of thought, the

technology is still been used, as it provides a lot of certainty in regard to the approach the chatbot is going to take when it receives input from a user. It of course has a lot of downsides. The fact that it can only function correctly when a rule is set, the door to not knowing how to respond is open, as it is very hard to predict the reaction of a user. This is why this type of conversational agents have to be very specific and developed very carefully, taking under consideration all that can go wrong in a conversation. The other big category of chatbots are artificial intelligence chatbots. This kind of conversational agents try to learn from the conversations they have. To do so, they use NLP(Natural Language Processing) together with some data set examples. The first chatbot of this kind was made by Joseph Weizenbaum, called ELIZA [48]. This chatbots, with enough training, should be able to detect conversational patterns, differentiate between different contexts and respond accordingly. Nonetheless, though all this traits seem perfect for developing a chatbot that emulates human conversation the best, it also comes with major disadvantages. It not only is a relative new line of investigation, but it also depends on strong computers that are able to manage all the processing of big enough data so that the chatbot is well trained. This may lead to the conclusion that even though chatbots have been around for a while, there is still no “perfect” conversational agent that offers the possibility of having a good conversation in an easy manner. It is true that science has provided many templates and ways in which chatbots can be configured without the use of code; making this technology very accessible to the public.

2.3 Implementation of chatbots in various fields

Chatbots, initially, were used with very simple objectives in mind as there was close to no NLP available. Nowadays, their use has expanded over many fields, like customer service, education, etc. Yet not many have explored the possibility of using chatbots as a tool for mental health patients. There are many ways of taking a survey digitally, but the use of chatbots can offer a new perspective. When Kim et al. [29] compared data from web surveys and chatbots, they concluded that overall, conversational agents can be useful. Since the use of chatbots encourages the interaction of the user, the answers where of a higher quality and more reliable.

At first one might ask if it is even useful to use chatbots in health

care, Elmasri et al. [14] developed a chatbot to address alcohol abuse by young adults. They tested their application with 17 participants aged 18–25 (10 male, 7 female) who passed a pre-screening to establish them as low to medium risk. In their conclusions, they studied the usability and effectiveness of the conversational agent and found that the chatbot had an overall positive response and people found it useful. Even though this paper was not directed with the specifics of mental health care in mind, it is relevant when considering chatbots in health care. Another example is that of Gillian et al. [6]. They developed a simple chatbot to propose a design for a conversational agent to be used in mental health counseling. Though the prototype of their conversational agent was simple, their paper concluded that the use of chatbots were rather useful, as the research showed that the users found the chatbot "safe" and easy to use. This proves also that the implementation of chatbots can help patients that feel uncomfortable with a face to face conversation. The study of Wang et al. [47] developed another chatbot oriented for smoker users. Their system was implemented to support pre-programmed information dissemination, health management and decision support. It was concluded that the system was cost-effective and showed that conversational agents are effective in triggering interactions for online groups.

So far it has been assessed that chatbots can be useful in health care when designed correctly. However, a big issue of implementing conversational agents is the means available. It is clear that the closer an artificial conversation emulates a human conversation the better; but sometimes the cost of achieving that goal is higher than the results received. Siddig et al. [42] designed and implemented PlyBot, a psychological self-help chatbot. PlyBot was a rule-based chatbot developed in a cloud based platform. They found the use of cloud based platforms very convenient for their work, as they provided many advantages for both users and psychologist. It is also true that they mentioned that using cloud-based platforms was sometimes limiting, as the lack of accessibility to the architecture and the fact that the code directly depends on the updates of the platform's documentation does not provide an ideal environment for development. Nonetheless, the fact that using this platforms enable users to access easily to the chatbot and help psychologist gather more specific, documented and safe information, makes up for the limitations.

Another example of a chatbot used in health care is "MedBot", developed by Rosruen et al. [41], it is a medical consultant chatbot developed using DialogFlow, a cloud-based platform. In their conclusions

they state that the system helped maximize the convenience of the user, as well as increase the capability of the system to provide service while decreasing the operational cost of medical consultant services. Further researches have received acceptance among healthcare professionals, like that of Kadariya et al. [27], who successfully prototyped a chatbot using Dialogflow and Firebase among other tools to develop a conversational agent able to answer questions, monitor their asthma relevant data and help them self-manage their asthma.

Chapter 3

Design

This section describes the architecture chosen to develop the digital chatbot, addressing all decisions made.

3.1 Requirements

The requirements needed to be fulfilled by the conversational agent are described using the MoSCoW approach, which consists of differentiating four categories: the Must-haves (requirements that the system simply must achieve), the Should-haves (even if this requirements are not met, the system will be valid, but they should be included), the Could-haves (these requirements could be part of the project if more time or resources were to be added) and the Will not-haves (requirements that will not be met but that could be developed in future updates). Following this line of thought, the requirements are as follows:

Must

- The system must be able to provide a questionnaire: the chatbot will need to be able to ask the user all the questions of the Zarit test interview, which will be explained in more detail later.
- The system must be able to save data and interpret it: all the data of importance sent by the user, this data being the answers to the questions, must be saved for future studies, but more importantly, the information has to be interpreted so the system knows how the quality of life of the user is developing and thus answer correctly.

- The system must provide feedback in the form of coping mechanisms based on their psychological assessment: this goes in line with the previous requirement, since the system will be able to interpret the data gathered, it must answer accordingly. Depending on the results of the Zarit Test, the chatbot will answer one way or another; providing useful tips to the user.

- The system must be deployable to other chat-based platforms: since all this project is focused on every-day users, it is only necessary that the platform is accessible to all; thus, the conversational agent must be deployable to a user-friendly platform.

- The system must be able to start a conversation if the user does not interact with the chatbot for a certain period of time.

- The system must be able to keep a short common conversation: for all it is worth, this project would not be interesting without the use of small-talk to simulate a normal conversation between 2 people. The conversational agent must be able to maintain some kind of interaction with the user other that the test questions in order to mimic closeness and concern.

Shoulds

- The conversational agent should interact with the user periodically: usually, the Zarit Test is conducted every now and then, causing the abandonment of the caregiver. This system should salvage that by being the one to start a conversation with the user if a certain period of time has passed without any interactions.

Coulds

- The system could use artificial intelligence and NLP to improve answers and be more dynamic.

- The system could interact with an external app used by health-professionals to help track the quality of life of the caregiver and make more accessible the medic-patient relationship.

Will Not Happen

- The system will not be able to interact with the outside world, for example to alert the authorities when feelings like suicide were detected.

- The system will not share information with a health specialist.

3.2 Architecture

Keeping all the requirements in mind, the time has come to choose what type of conversational agent will be implemented. Currently, we could divide the chatbots into two categories, artificial intelligence bots and ruled based bots. Artificial Intelligence Bots are much more complex but also more dynamic than rule-based chatbots. AI bots can either use machine learning techniques or NLP (Natural Language Processing) to make decisions, aiming to seem more human-like. With these methods not only can you simulate a human conversation better, but you can also learn from mistakes and inputs from the user. On the other hand, we have Scripted chatbots, very simple making but very limited results since they work under an “if/else” philosophy.

On a first glance it seems obvious that the way to go is AI chatbots considering that we are aiming to design a conversational agent that shows emotion, and closeness. However, the time needed to implement correctly an AI chatbot is big, hard to predict and the infrastructure needed to do so is out of reach. But here comes the alternative to both of these options: Off-the-shelf cloud platforms. Cloud based chatbot development platforms provide service up to a limit for free. They offer advantages on the following categories:

- **Ease of use/Customizable design:** simple and very intuitive to implement.

- **Data Storage:** recollect and save data for later usage without spending more money than needed on personal storage-devices.

- **Security and Ethics:** since it is a public cloud-based product it is forced by law that security and integrity policies are followed.

- **Cost:** very economically attractive, since the magnitude of this project will not need a lot of space.

- **Scalability:** the system can scale on-demand and grow or diminish when needed.
- **Compatibility:** most of these services have implemented the option to deploy projects to other platforms such as Whatsapp or Messenger.
- **Technology:** some of the platforms that provide this service use some kind of NLP, which would otherwise be out of reach.

In figure 3.1 the architecture designed for this project is displayed. The system can be divided into three layers, the presentation module, the chatbot module and the data storage module.

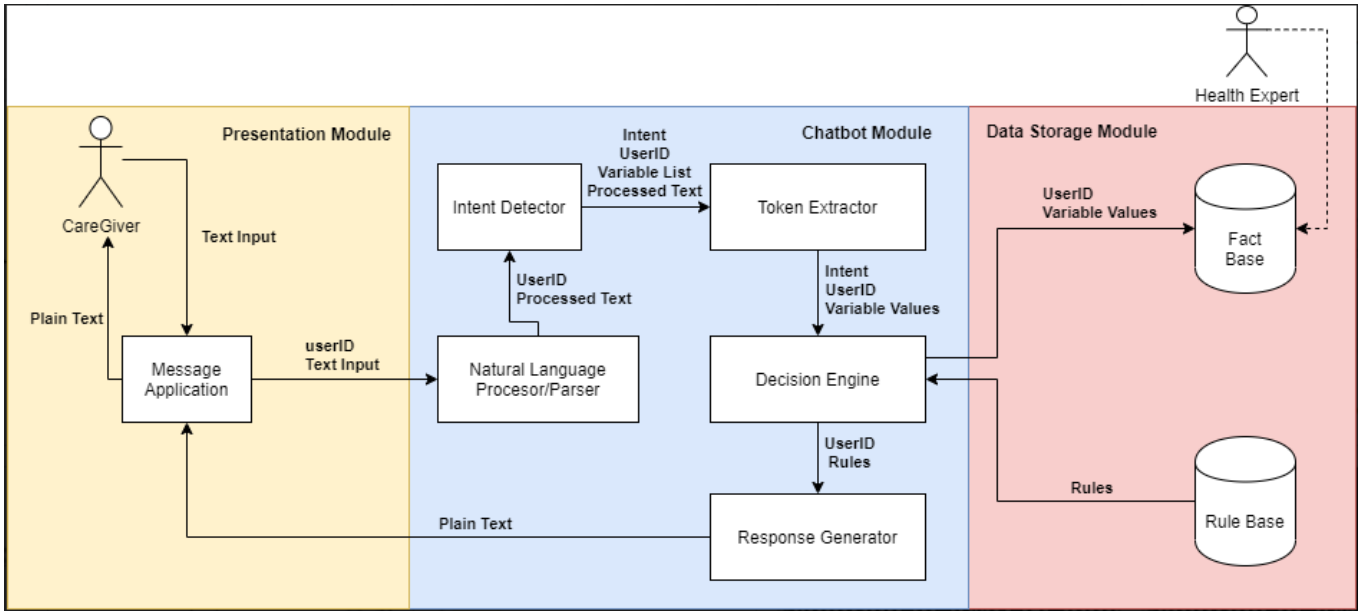


Figure 3.1: Architecture Design.

The user will send a text input to the message application. This application will send to the Chatbot Module the userID together with the text input. This id is univocal to the message application, so it can be used as the univocal identifier for each user within the data layer module. The data will be processed by natural language processor (NLP), at this stage punctuation and empty words such as pronouns will be erased, sending a processed plain text to the intent detector. The user's id is also sent. Once the intent is recognized by the intent detector, a list of variables such as X,Y and Z needed will go through the token extractor.

The token extractor is in charge of identifying this variables within the processed text. Through the flow, both the userId and the intent are maintained. When all the data necessary is gathered, the variable list, the userId and the intent are forwarded to the decision engine, which will decide what kind of answer the user should receive. The different type of answers include mainly three categories, the "final answers" which represent the end to a conversation, answers perceived as small talk or answers related to the Zarit Test Burden Interview. The decision engine will be accounted for saving data in the knowledge base in the data storage module as well. This information could be available for health experts to analyze easily. Once the decision engine has selected an option, the rules that identify the type of answer and the userId are sent to the response generator. The response generator will act according to the rules received and will send the information the the user with the userId registered. Whatever the response that is sent, it will be sent to the message application as a plain text, for the message application to renderize such text in the correct form.

3.3 Use cases

There are two main use cases when it comes to initiating a conversation:

The diagram displayed in scenario A, where the user is the one to initiate the conversation, would be the following:

The use case A, as figure 3.2 depicts, will start with the user being the first to interact with the phone app in the form of a text or invoice message. This message will later be sent to the chatbot through a JSON file. The chatbot will do the internal processes necessary and respond to the mobile app, message that will be received and read by the user.

Use case B would be the chatbot initiating the conversation (figure 3.3). Figure 3.3 is very similar to figure 3.2, but in this scenario, it is the chatbot which starts the conversation by sending a message to the mobile phone app, the app receives the message and presents it to the user in a text form, the user then answers and this response is sent back to the chatbot.

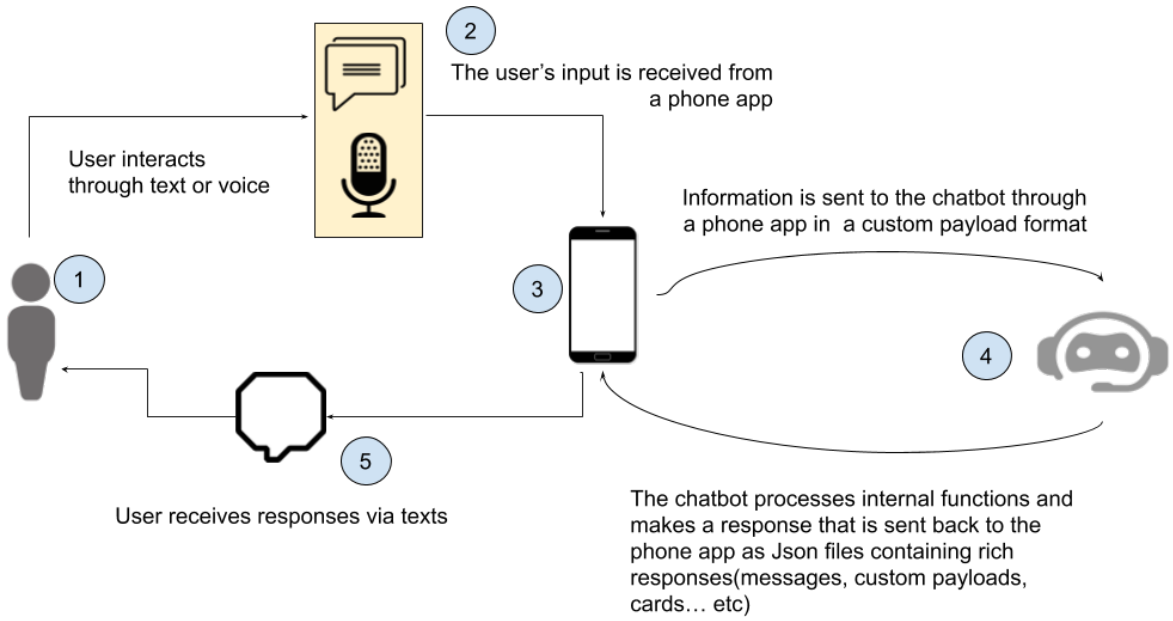


Figure 3.2: Use case A: The user initiates the interaction.

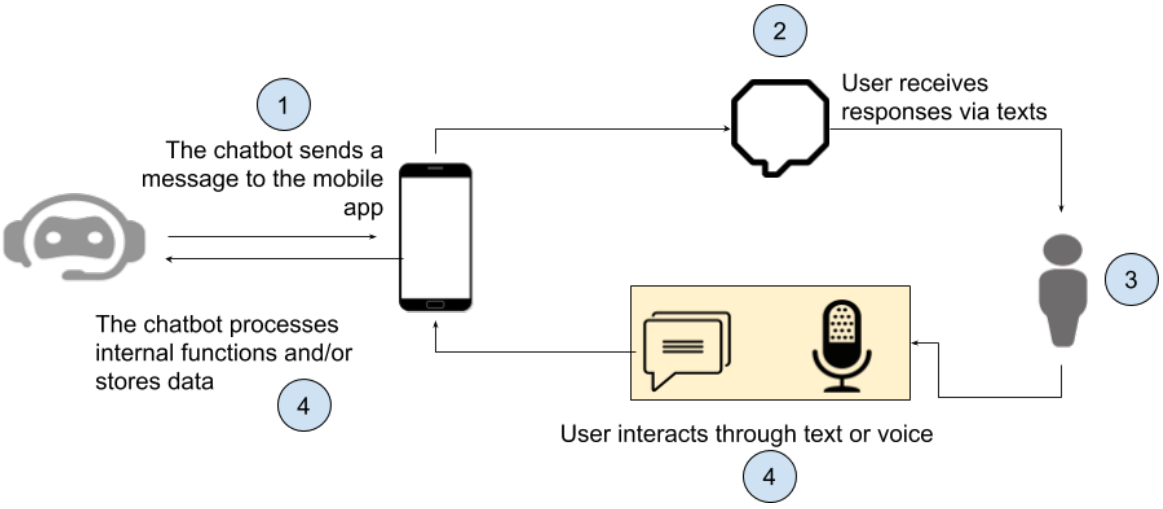


Figure 3.3: Use case B: The chatbot initiates the interaction.

Chapter 4

Implementation

After much thought and analysis of all the options, it was decided that the most adequate for this application would be to use cloud based platforms for the implementation. The use of cloud based platforms offered the main advantage of easy deployment to other platforms as well as the mix of both NLP and rule-based chatbots. Furthermore, the use of these type of tools could be set as a starting point for other researches to take the task further into development.

4.1 Tools

There are many cloud based platforms to choose from, IBM Watson or Azure Bot Service are just some of them. After exploring the different alternatives, it was decided to use DialogFlow. DialogFlow is a service provided by Google for the development of chatbots. It offers many advantages, including that of easy deployment to other platforms and the internal use of NLP, though limited. IBM Watson was also heavily considered, but rejected after comparing both interfaces and services provided within the free plan.

Another aspect that needed to be taken under consideration was the message application that would be used. Dialogflow offered an easy deployment to 14 different platforms, but due to time limitations only one was selected: Telegram. Telegram is a well known message app that is both available for free in IoS and Android, making the internet connection the only requirement needed for users. It is also quite popular, as

according to the U.S. Securities and Exchange Commission, the number of monthly Telegram users as of October 2019 is of 300 million people worldwide.

For production hosting, Cloud Functions for Firebase will be used, as DialogFlow is specifically designed to work combined with Firebase, although other environments could be used as well. This was also one of the reasons to choose DialogFlow. Since it is meant to be used with Firebase, and Firebase at the same time offers both the API to code all our functions and data storage, including that of real-time databases; it makes the job much easier and ensures a good connection between all elements. To illustrate the work flow that the system is going to follow a diagram has been made showing the different steps:

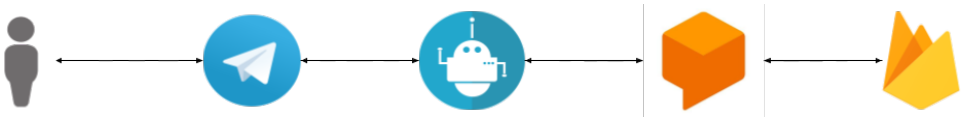


Figure 4.1: Communication Flow Chart of the system's elements.

As image 4.1 depicts, the user via Telegram will communicate with the bot, which is directly connected with DialogFlow. DialogFlow will then use Firebase to fulfill intentions and store data. In this step is where functions like result evaluation and metrics will be calculated and then returned to DialogFlow to show through the bot into Telegram; thus, the double direction arrows.

4.1.1 DialogFlow

DialogFlow, as explained above, is a cloud-based platform for the development of digital conversational agents. DialogFlow works with agents, which in our case of study are the chatbots, so from now on, agent is short for digital conversational agent. DialogFlow works with many components, but these are the most important and relevant to our project:

- **Intents:** according to the official DialogFlow documentation, an intent *"categorizes an end-user's intention for one conversation turn"*. In other words, an intent represents what the user wants to achieve when conversing with the bot, or seeing it the other way around, it is what the bot is expecting to recognize when interacting with the user. An example is shown in Figure 4.2.

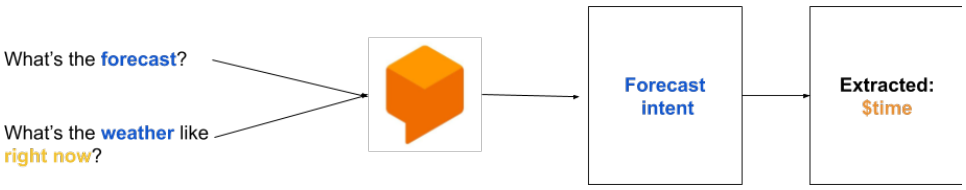


Figure 4.2: Detection of basic intent.

Figure 4.2 shows a case scenario where a user asks for the weather, also known as `forecast`. DialogFlow detects the intent Forecast. DialogFlow can also define intents so they can extract useful information from the user's input, in this case DialogFlow extracts time. This information can later be used to perform a weather query for the end-user.

- **Contexts:** DialogFlow sometimes needs to be provided with a context to handle interaction with the user. To set an example of a context, if a user answers to a yes/no question and DialogFlow has set two possible answers for a "yes" input:
  - "Oh that is great."
  - "Oh that is horrible."
- **Entities:** as the official DialogFlow documentation [20] states, *"Dialogflow provides predefined system entities that can match many common types of data. For example, there are system entities for matching dates, times, colors, email addresses, and so on"*. For example, if a user enters the text *"let's see the movie at 22:00"*, in this case the time, 22:00, is an entity.
- **Fulfilments:** fulfilments are used to provide a more dynamic response. Fulfilments are linked to intents, so when an intent has an enabled fulfilment, DialogFlow responds by calling the service associated with it. Here is when FireBase comes into place. A simple scenario would be a user who wants to make a reservation for dinner at a restaurant on Friday night at 12:00; the service can check the database to check availability and then respond to the user's request. Each intent has a setting to enable fulfilment. For our architecture design, when an intent with fulfillment is matched, DialogFlow sends a request to the webhook service connected to Firebase. Then, the service performs the action needed and the service sends back a webhook response to DialogFlow. Figure 4.3

is an image taken directly from the DialogFlow documentation, depicting the fulfillment flow explained earlier.

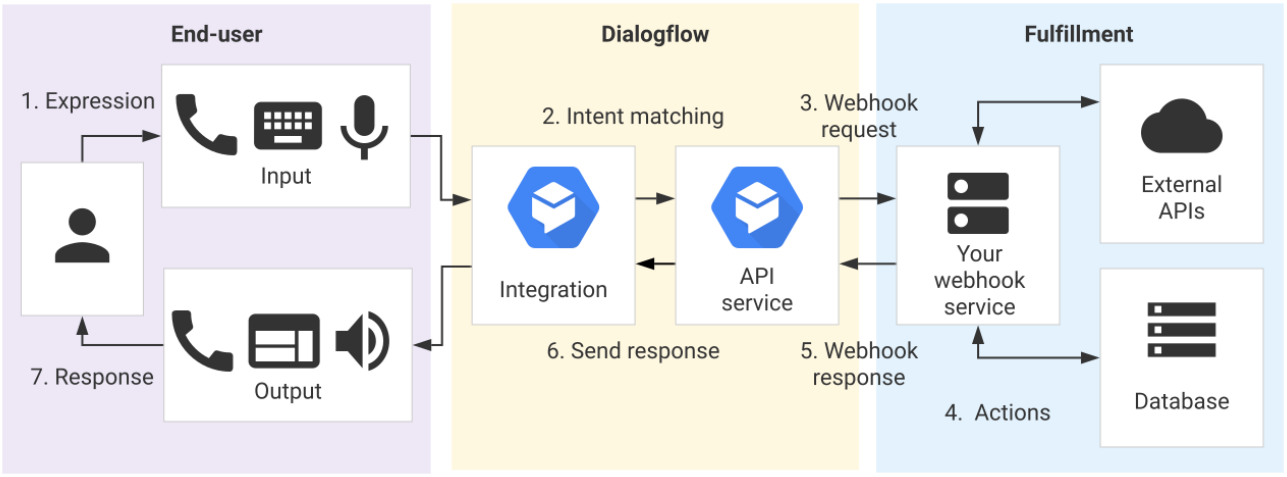


Figure 4.3: Processing flow for fulfillment. Reprinted from [21]

4.1.2 Firebase

According to Wikipedia, Firebase is a *"Mobile and web application platform"*. While this is true, for our case of study we are going to use other functionalities of Firebase, the cloud functions to be more precise. The official documentation of Firebase says that *"Cloud Functions for Firebase lets you automatically run back-end code in response to events triggered by Firebase features and HTTPS requests. Your code is stored in Google's cloud and runs in a managed environment. There's no need to manage and scale your own servers"*. When talking about interaction between DialogFlow and Firebase, the developer can code a Cloud Function deployable to each par. Firebase offers two cloud-based, client-accessible database solutions that support real time data syncing:

- **Realtime Database:** an efficient, low-latency solution that requires synced states across clients in real time. It works with a single JSON file that stores all the data, so hierarchical data is harder to organize at scale.
- **Cloud Firestore:** it is based on the realtime database but with a more intuitive data model. It stores data as collections of docu-

ments, so complex data is easier to organize and scale and does not require so much data preparation.

4.1.3 The Zarit Test

As mentioned earlier, the Zarit Burden Interview is a test that measures the level of burden experienced by caregivers of patients with dementia. The revised version of this test consist of 22 questions. Each question is a statement which the caregiver is asked to endorse using a five point scale; where the scale runs from zero (Never) to four (Nearly Always). Some examples of these questions are: *"Do you feel that your relative asks for more help than he/she needs?"* or *"Do you feel embarrassed over your relative's behaviour?"*. Considering the scenario for the performance of the Zarit Test Interview, it comes as no surprise that the data gathered is weak. Usually, not only is the test not taken frequently, but also, since it is so long, the attention given to the questions diminishes in time. This process also is limited by the moment to which it is taken. People are not machines, we have our good and our bad days; and a lot of the times, a diagnosis is made not considering the specific circumstances of that day or time. With the use of chatbots, the test can be delivered time at a time; and the ending result will be more fair. For example, the first day the bot will ask the user questions 1 to 6. The next day, questions 7-15 and so on; that way, the test can be responded under different circumstances and the user will not get too tired. This also helps with the fact that some questions are very hard and personal to answer, and having to face them all at once affects the caregiver negatively.

The interpretation of the final score is fairly easy, when all the points have been added. If the score runs between 0-21, the user suffers from little or no burden. If it is between 21-40, a mild to moderate burden is detected. From 41-60, the caregiver suffers from moderate to severe burden and last but not least, ranges between 61-88 indicate severe burden of the caregiver. As mentioned earlier, depending on the results, the chatbot will answer one way or another; following the guidelines of Cooper and peers [11]. It also needs to be mentioned that since the complete scores of the test may take a few days, meanwhile, the bot will offer motivational quotes as part of the everyday conversation with the caregiver, or helpful information along with small talk to make the task less artificial. The next table includes all of the Zarit test questions.

Question		Score
1	Do you feel that your relative asks for more help than he/she needs?	0 1 2 3 4
2	Do you feel that because of the time you spend with your relative that you don't have enough time for yourself?	0 1 2 3 4
3	Do you feel stressed between caring for your relative and trying to meet other responsibilities for your family or work?	0 1 2 3 4
4	Do you feel embarrassed over your relative's behaviour?	0 1 2 3 4
5	Do you feel angry when you are around your relative?	0 1 2 3 4
6	Do you feel that your relative currently affects your relationships with other family members or friends in a negative way?	0 1 2 3 4
7	Are you afraid what the future holds for your relative?	0 1 2 3 4
8	Do you feel your relative is dependent on you?	0 1 2 3 4
9	Do you feel strained when you are around your relative?	0 1 2 3 4
10	Do you feel your health has suffered because of your involvement with your relative?	0 1 2 3 4
11	Do you feel that you don't have as much privacy as you would like because of your relative?	0 1 2 3 4
12	Do you feel that your social life has suffered because you are caring for your relative?	0 1 2 3 4
13	Do you feel uncomfortable about having friends over because of your relative?	0 1 2 3 4
14	Do you feel that your relative seems to expect you to take care of him/her as if you were the only one he/she could depend on?	0 1 2 3 4
15	Do you feel that you don't have enough money to take care of your relative in addition to the rest of your expenses?	0 1 2 3 4
16	Do you feel that you will be unable to take care of your relative much longer?	0 1 2 3 4
17	Do you feel you have lost control of your life since your relative's illness?	0 1 2 3 4
18	Do you wish you could leave the care of your relative to someone else?	0 1 2 3 4
19	Do you feel uncertain about what to do about your relative?	0 1 2 3 4
20	Do you feel you should be doing more for your relative?	0 1 2 3 4
21	Do you feel you could do a better job in caring for your relative?	0 1 2 3 4
22	Overall, how burdened do you feel in caring for your relative?	0 1 2 3 4

Table 4.1: Zarit Test Interpretation of Score: 0-21 little or no burden 21-40 mild to moderate burden 41-60 moderate to severe burden 61-88 severe burden

Group	Group A Social Life	Group B Money/Means	Group C Expectations	Group D Personal Values
Question id	3,6,12,13	15,16,19,7	1,8,14,18 19,20,21,22	2,4,5,9 10,11,17

Table 4.2: Zarit Test questions divided into 4 groups.

For the purpose of organizing the conversation and giving it a little bit of context, the questions from the Zarit test have been divided into 4 categories, represented in figure 4.2. Please note that these categories are completely subjective.
When a conversation is taking place, the chatbot will first ask all the questions of one category, and when it is done it will randomly select another category, until all questions have been responded. The reason for this division of questions into categories has being made to give a little bit of continuity to the conversation too. For example, if the bot is asking the user about hobbies or friends, it would make more sense to ask questions about their social life, rather than a question regarding money.

4.2 Development

The first step, naturally, was to create an agent in DialogFlow. This was easily done following the instructions and the official documentation. If you have created the agent successfully, a panel like the following should appear.

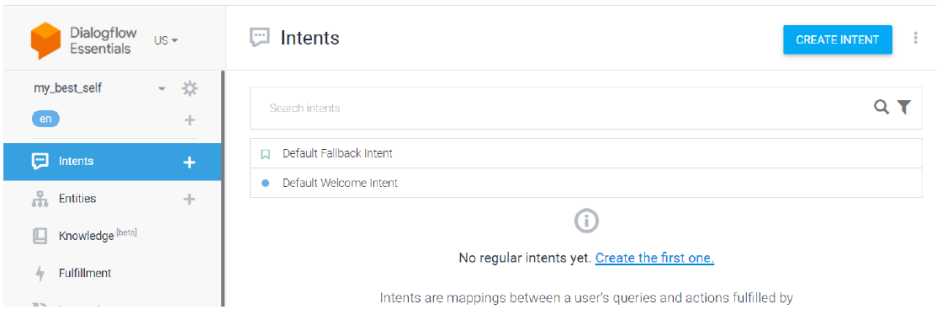


Figure 4.4: User Interface of a newly DialogFlow agent.

The panel is very intuitive, on the very top we have the name of the agent, in our case tester. Following we have the language settings,

and after that the intents, entities, etc. By default, the "Default Welcome Intent" and the "Default Fallback Intent" are created. The welcome intent comes with the Welcome Event already set up; it is pretty self-explanatory. The fallback intent is used when the agent does not recognize any intent from a user's input. If the developer selects an intent, say the Welcome intent, the interface shows various options, like for example the option to set a specific context; this feature will be used many times along the project. Events can also be configured, as can be training phrases, action and parameters, and responses. But probably the most important feature is found at the end of the interface, and that would be the option, that if clicked, the user can decide whether to activate a fulfillment or not. If the user chooses to enable a webhook for this intent, DialogFlow will manage the intent, in our case, the inline editor; space that enables easy development for firebase functions. In the fulfilment section, the user can enable or disable the inline editor; at this point, if it has not been done earlier, DialogFlow will ask you to activate the project in Google Cloud Console; this is mandatory. It also mentions that you have to choose a billing plan, but do not get confused, the free plan is a type of billing plan, in fact it is the plan that was used for some time in the project. Unfortunately, a new billing plan had to be paid.

Once the project has been initiated in Google Cloud Functions, access is granted to the inline editor. DialogFlow offers two files that will be deployed to firebase when needed; both files are written in JavaScript. First, the index.js file is where functions will be developed, while the package.json contains all the libraries necessary for the correct functionality of the code. Having these libraries updated is of the utmost importance, since there are features used in this paper that only correspond to the latest versions. DialogFlow is very conservative in this matter and does not configure the latest version of libraries as a default option, nor does it come with all the libraries needed.

```
"dependencies": {
 "actions-on-google": "^2.5.0",
 "firebase-admin": "^8.2.0",
 "firebase-functions": "^2.0.2",
 "dialogflow": "^0.6.0",
 "googleapis": "^27.0.0",
 "dialogflow-fulfillment": "^0.6.1",
 "request": "^2.83.0"
}
```

Listing 4.1: Desired package dependencies inside the package.json file

In our case, we have followed the configuration depicted in the code 4.1. On the dependencies section, one of the must-libraries has to be the dialogflow-fulfillment. Thanks to this library we can send different types of responses, like cards, images or payloads; the latest being extremely important for a dynamic conversation style. This aspect will be viewed in more detail later on. Another extremely important library is that of firebase-functions and firebase-admin; without these libraries the correct deployment to Firebase will not be successful, nor will be the calls to the database. At risk of sounding repetitive, the version of the dependencies is extremely important.

Once we have everything setup in DialogFlow, the time has come to connect the DialogFlow project to Firebase. When we first enter the Firebase console we see the option to add a new project. It is of the utmost importance that you log in with the same google account that you have used for DialogFlow, since if you have configured the project correctly into Cloud Functions, Firebase will automatically give you the chance to link the new Firebase project to an existing Google Cloud Platform project, making your life much easier. The next step is to check that the configuration is correct, with that purpose in mind we can check all the setting details in the tool icon placed on top of the sidebar menu. When clicked, we see an interface with the general settings of our project, where we can find the name of the project, the ID and the API key. The next step to properly check that the configuration between Google Cloud Functions and Firebase has gone smoothly, is to go to the user interface of Google Cloud Functions of our project, then click on the Navigation Menu > APIs and services >API credentials we should see the same data as in Firebase. Going back to Firebase, to setup a database click on the section Database. We have the option of creating a Cloud Firestore or a RealTime database. At first, I opted for a RealTime database, and though it works fine, I had some issues with the version of dependencies and the connections so it was later decided that Cloud Firestore was a more stable product, as RealTime Database is a recent launch of Firebase.

Cloud Firestore is a NoSQL database, meaning that the concept of tables is not applied. The database is organized in collections, each of these collections has an amount of documents (JSON). Documents within the same collection do not have to have the same structure, but for the

purpose of organization, in our case, each document has the same structure as another of the same collection. For starters, we have created the user collection, where each file contains the basic information of a user, like the name, the name of the relative they care for, age, etc. Another four collections have been created, one for each category of Zarit Test questions, displayed in 4.2. Inside each category, each document corresponds to a question. The intention of this system is to introduce the test questions in an informal conversation. For that reason, we have created another 4 collections, one for each group of Zarit Test questions. This collections have everyday life questions or statements. For example, if the chosen group is A, Social Life, one example of an introductory question to the Zarit Test items might be *"have you done anything fun lately?"*.

Summarizing, the Firebase database will have eight initial collections; collections that run from QGroupA to QGroupD and a user collection. Finally, to have a working link of Firebase to Dialogflow, it is necessary to setup the following parameters inside the inline editor, in the beginning of index file, located in Dialogflow under the tab of Fulfillment.

```
1 'use strict';
2 const functions = require('firebase-functions');
3 const admin = require('firebase-admin');
4 const {WebhookClient} = require('dialogflow-fulfillment');
5 const {Card,Suggestion,Payload} = require('dialogflow-fulfillment');
6 admin.initializeApp();
7 const db = admin.Firestore();
```

Listing 4.2: Code needed to connect the Inline Editor to the Firebase database

Line 2 of listing 4.2 is calling the functions of `firebase-functions` , that should have been included in the package.json file as shown in listing 4.1. This will enable the developer to call the functions of Firebase. Line 3 establishes a variable only used for admin-related functions, like for example, granting access to the database, as shown in line 8, where a const variable called db will be used in the https calls to the database. Admin is also in charge of initializing the app, thus line 7 of the code. Without these lines of code, the inline editor would not be able to interact with the database, so they are of the utmost importance. Line 5 calls for objects of Dialogflow, later used to make a dynamic answer with the user.

Now that the connection of Firebase to Dialogflow has been made, it is time to finalize the setup of our environments by connecting Telegram to Dialogflow. It is not necessary to connect Firebase to Telegram since these tools will not be interacting directly with one another, all communication will take place through Dialogflow, as depicted in figure 4.1. To do so, going back to the Dialogflow interface, under the tab of "Integrations" on the left side panel, a lot of options are found under different categories, like for example Telephony or Open Source. The section of interest is text based; where apps like Whatsapp or Messenger are found, including Telegram. To connect to the platform, in this case Telegram, enable the toggle button as shown in figure 4.5.

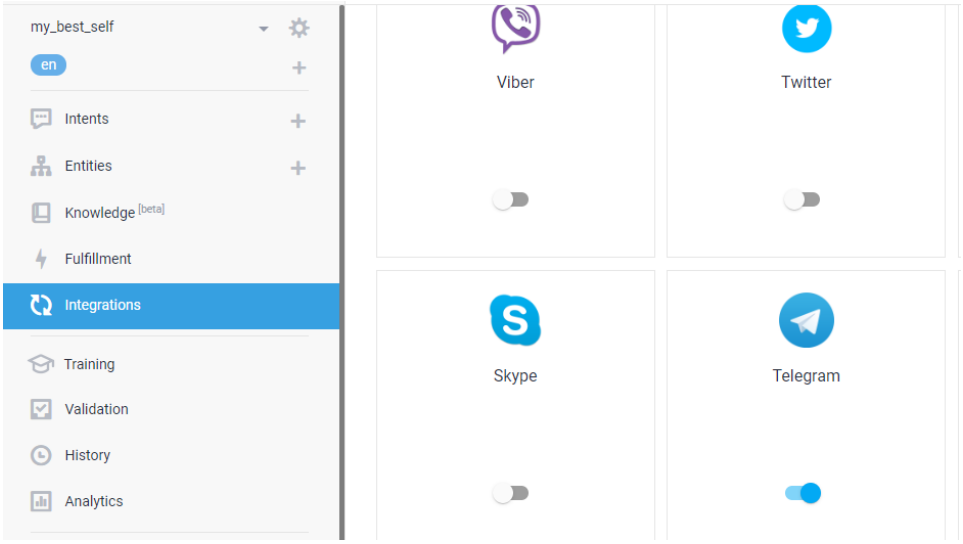


Figure 4.5: Applications enabled to an easy link to Dialogflow.

A pop-up dialog as the figure 4.6 will appear. To get the token that is being asked for we have to go to Telegram and interact with the BotFather. To initiate the process the command /start must be sent, instantly, the bot will answer with a list of different commands that can be used to set up our bot (images 4.7 and 4.8). The next step is to send the /newbot command, that indicates that you want to start a new chatbot within Telegram. The FatherBot will in that instant ask you for a name, in our case MyBestSelf and for a username MyBestSelfbot. As the instructions indicate, the username has to end with the word bot. It will be this username the token used for users to get the chatbot functioning in their accounts.

If everything is correct, the FatherBot will give you the token access

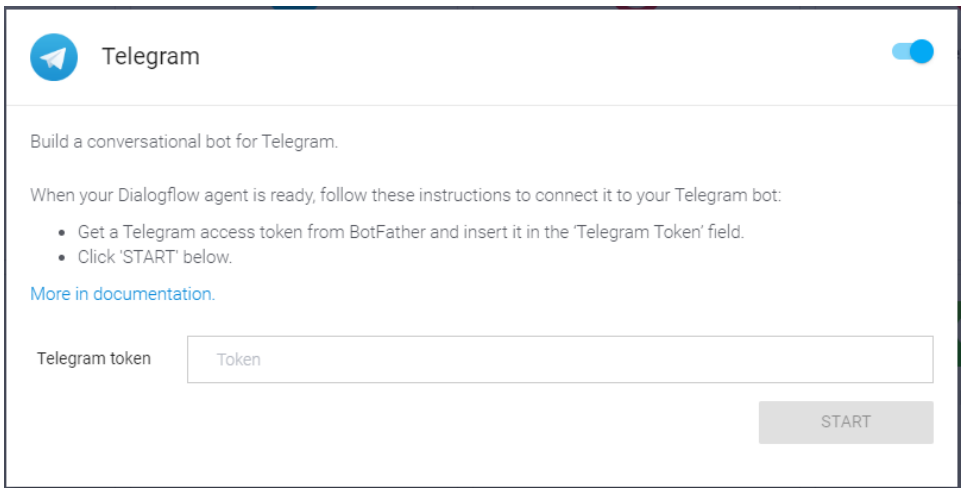


Figure 4.6: Pop up of Dialogflow when Telegram enabled.

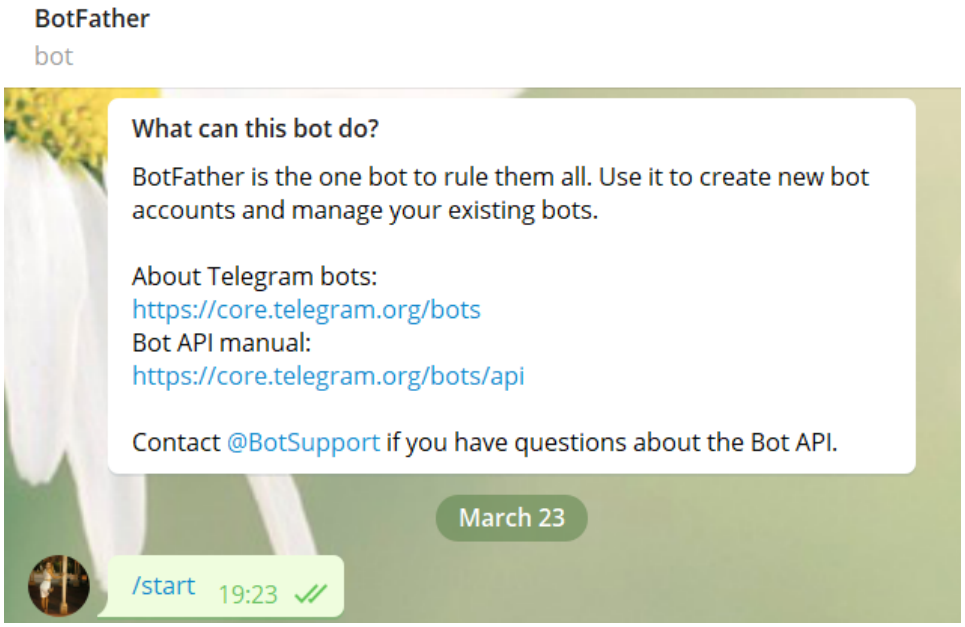


Figure 4.7: First step in creating a Telegram chatbot.

that is the token asked by DialogFlow earlier; all needed to be done is copy paste that token in the DialogFlow pop up and that way a functioning Telegram bot will be linked to DialogFlow. Once everything is set up, coding can begin.

Figure 4.8: process of creating a Telegram chatbot.

The first step is to detect a user. The first time a caregiver interacts with the chatbot, the system should be able to detect that the user in question is included in the database or not. This also implies that every time a user greets the bot, the system must run the same function and check for the existence of the user's id in the database.

Following this line of thought, the first thing that needs to be done is to establish the Welcome Intent. DialogFlow already has set up this intent, so changes need to be made. First it is necessary to enable webhook call for this intent as shown in figure 4.8. Reminder: this feature can be found at the end of the user interface of DialogFlow of each intent.

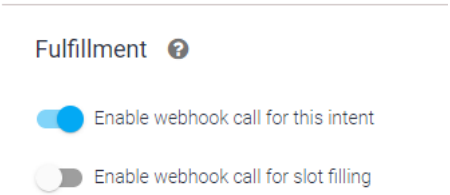


Figure 4.9: Enabled webhook call for an intent.

Once done, in the Inline Editor we have to map the Welcome Intent to a function, this is done by setting a so called intentMap. Usually this is done at the end of the index.js file. This has to be done for every intent that functions as a filfillment.

```
let intentMap = new Map();
let intentMap.set('Default Welcome Intent', welcome);
let intentMap.set('Default Fallback Intent', fallback);

agent.handleRequest(intentMap);

```

Listing 4.3: Mapping of intent to a function.

The code shown in listing 4.3 is already implemented in the inline editor the first time we enable the webhook, and it will be taken advantage of. The next step is to find/create a function with the exact same name written in the intentMap handler, in our case the function will be welcome. Also notice that the name of the intent has to match exactly the name given in the Intent interface, as if it is not absolutely accurate, the association to the function will not be made. The function Welcome is fairly simple. This function, by the set up, will always be called when a user says “hi” to the chatbot.

```
function welcome(agent){
  agent.add(`Welcome`);
}
```

Listing 4.4: Welcome function.

Agent.add() will display whatever it is inside the parentheses in the Telegram interface. So in this scenario, for starters, the bot will always answer ‘Welcome’ to the user, as long as the welcome intent is detected. Listing 4.4 is the basic configuration of the welcome intent, but that doesn’t mean it can be modified. In our case, the welcome function has to be able to identify whether the user exists on the database or not, and act accordingly. To do so, the following logic has been implemented inside the welcome function.

```
const userId = request.body.originalDetectIntentRequest.payload.data.from.id.toString();
const first_name = request.body.originalDetectIntentRequest.payload.data.from.first_name;
return db.collection('users').doc(''+userId).get()
    .then((doc) => {
        if (!doc.exists) { //start get-to-meet conversation
```

```
6       }else{//ask small talk question
7       }
```

Listing 4.5: Welcome function.

In the first two lines, the username and the userId are extracted; following is a call to the database, specifically to the "userId" document inside the "users" collection. If the document doesn't exist, the get-to-meet conversation will take place, but if the user does exist, the chatbot will directly start asking small talk questions.

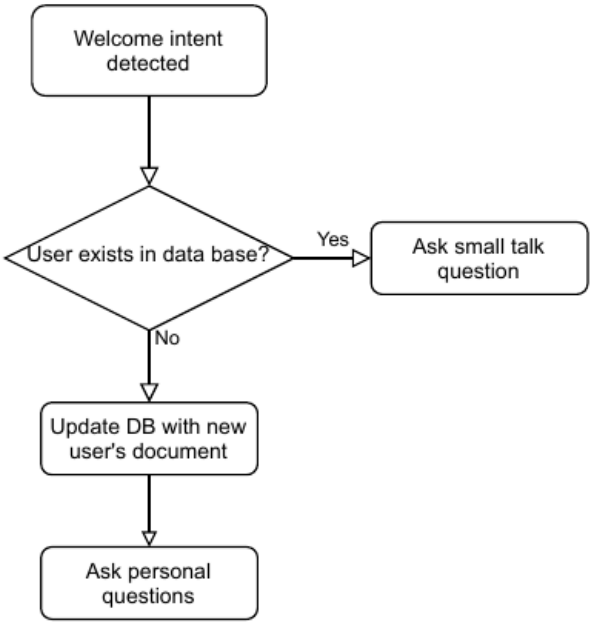


Figure 4.10: Logic inside the welcome function.

4.2.1 Get-to-meet conversation

The first conversation a user has with the chatbot is always the same for all users. Simple questions are asked with the intention of giving a little bit of reality to the conversation; but also to gather data that will later be used throughout the conversation. It is very important to understand the fundamentals of context from this point forward. Contexts, as said before, define the general purpose of a conversation. When a developer defines a context, it can not only choose the name of the context, but also the lifespan, meaning, for how long that context is going to be enabled.

Contexts can be used also to set variables private to each user; this is as necessary as it is helpful since by saving the user's data inside contexts, we can have various users using the system at the same time, without the information getting messed up. We could also achieve this by calling the database every time personal information is needed, but by the use of contexts, we save time, calls to the database and ensure that the information is right, private to each user, and available. To use a context in such a way, we must give it a long lifespan, so the context is available through the whole conversation. the following code defines a context called *contextvariable* that will be used throughout the duration of the conversation; it will be referred to as "the general context" from this point forward.

```
agent.context.set({
   'name':'contextvariable',
   'lifespan': 100,
   'parameters':{
     'userID':userId,
     'religion':'none',
     'currentQ':'',
     'currentQuestionID':'',
     'GA':[],
     'GB':[],
     'GC':[],
     'GD':[1,2,3,4,5,6,7],
     'zaritQuestionAsked':false,
     'currentSMGroup':'D',
     'notDone' : [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22]
     }
});
```

Listing 4.6: Setting a general context with a large lifespan.

This context will be private to each user, taking into account that the varaible userId is the one extracted in 4.5. To save the userId is very helpful since throughout the conversations many calls to the database will be made, and many of those will be to the document of that specific userId. Another parameter worth mentioning is *notDone*. This parameter has the id of the Zarit Test questions that have not been answered by the user; since this is the first time the user is speaking to the chatbot, naturally *notDone* contains all 22 questions. If this context was defined with a known user, it would be setup differently. *GD* also has some information, this parameter represents the small talk questions of Group D(table 4.2). Since this is the first time this user is interacting with the system, there is no document with its user id. To create such document the following code in listing 4.7 has been implemented.

```
agent.add(`Well hello!! It is great to meet you, what is your name?`);
agent.setContext({name: "firstTimer", lifespan:2});
var data={
    'id':userId,
    'name':first_name,
    'doneQuestions': [],
    'answers':{},
    'notDone':[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22]
};
return db.runTransaction((dataDB)=>{
  dataDB.set(db.collection('users').doc(''+userId), data,{merge:true});
  return Promise.resolve();
}).catch((err) => {
  console.error(`Error creating file: `+err);
});
```

Listing 4.7: Setting a general context with a large lifespan.

To successfully create a document the function `set` is used. The parameters in this case are the name of the document together with the collection and the data that we want to have inside such document. The `merge:true` parameter is used to insert the new document inside the collection that already exists without deleting the other documents. In this section of code an agent.add() has also been added; this is what the system will send back to the user when the promise has been resolved and returned. If the promise doesn't resolve, the agent.add() statement will never be sent. The fact that the document is first created doesn't mean that other fields can't be added later on. The creation of a new document with for example, id 123456, should look something like figure 4.11.

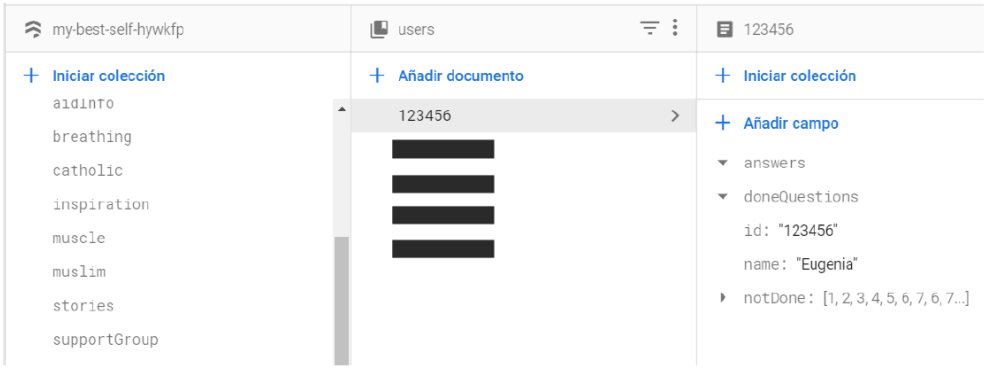


Figure 4.11: New document inside the collection users with the id of 123456.

As code in 4.7 shows, the agent will ask the user for its name. This

question will be followed by a number of other questions like age, name of sick relative (4.12) or religion(4.13).

Figure 4.12: First conversation between agent and user. Part1.

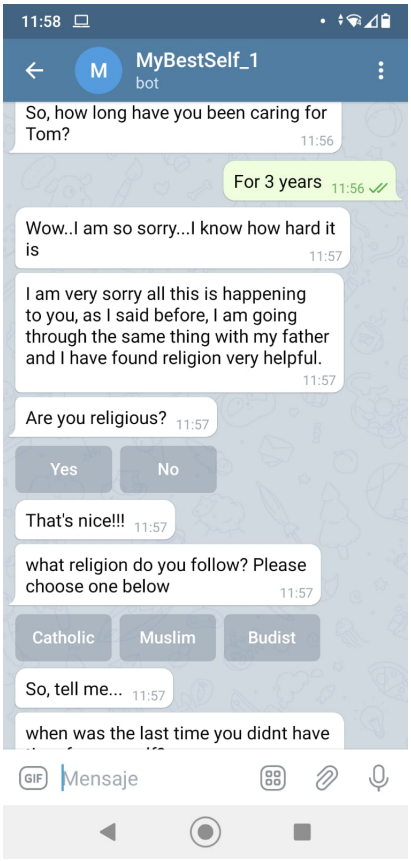


Figure 4.13: First conversation between agent and user. Part2.

Even though in figure 4.13 it seems that the user has not answered, he/she has in fact clicked a button, in the scenario of the figure it has said he/she is religious and chosen the catholic faith. This information can be later used along the rest of the conversation and it is also saved in the document of the user. Each time the system has to save some personal information that is for sure going to go to the user's document as an independent field, the function `updateProfileData`, portrayed in listing 4.8 is called. The async function uses again `set`, it is also very important to set the *merge:true* parameter, since if not done, all the information inside the document could be overwritten. As for parameters, the function receives the name of the new field (keyP) and its value.

```
async function updateProfileData(agent, keyP, value){
    let user = agent.context.get('contextvariable').parameters.userID;
    const cityRef = db.collection('users').doc(''+user);
    const res = await cityRef.set({
      [keyP]:value
    }, { merge: true });
  }
```

Listing 4.8: Saving personal data inside the corresponding document.

The question about religion is the last "get to know" question, as when the system saves the information, it starts the process of asking small talk question; process explained in the next sub section. It is important to mention in this part of the conversation when it is needed to add new information to the main context, due to DialofFlow's configuration, the context must be overwritten. This is done in the function *updateContext*, the function receives the name of the field and its value, then, the parameters of the context are saved on a local variable. Inside this variable, the new field and value are added and then the context is overwritten with the new parameters. A context is interpreted as a JSON file, so it is fairly easy to navigate through its structure.

```
function updateContext(field, value){
    let contextVariables = agent.context.get('contextvariable');
    contextVariables.parameters[field] = value;
    let sessionContext = {
      'name':'contextvariable',
      'lifespan': 100,
      'parameters':contextVariables.parameters
    };
    agent.context.set(sessionContext);
}
```

Listing 4.9: Adding new information to the main context.

From this point forward, every time the user greets the agent, it will enter the second logic of the welcome function (listing 4.5).

4.2.2 Conversation with a known user

Since the document for the user is already created, it is only necessary to set the context with the values of the document. The process is very similar to that of listing 4.6, with the only difference that the values of

the parameters come from the user's document in the database (listing 4.10).

```
const data = doc.data();
let doneQ = data.doneQuestions; //preguntas del test de Zarit hechas
let notDone = data.notDone;
agent.context.set({
    'name':'contextvariable',
    'lifespan': 100,
    'parameters':{
        'userID':userId,
        'religion':data.religion,
        'currentQ':'',
        'currentQuestionID':'',
        'GA':[],
        'GB':[],
        'GC':[],
        'GD':[],
        'zaritQuestionAsked':false,
        'currentSMGroup':'',
        'notDone' : []
    }
});
updateContext('notDone',notDone);
agent.add(`It is great to see you again!`);
```

Listing 4.10: Setup of main context when the user is already defined in the database.

Although the rest of the conversation is full of random responses to break predictability, there still exists a general logic to follow. Now that the user is registered in the database, it is time to start asking small talk questions that will serve as an introduction for the Zarit Test questions. Ideally, there would be plenty of small talk questions so that the agent would not repeat the same questions every time it interacts with a user, but there should be at least the same number of small talk questions as there are in the corresponding Zarit Test Group. Taking this under consideration, there are 2 types of questions/statements within small talk questions, the *yes/no* format or the *any* format. The first kind are questions that require a yes/no answer to be detected correctly, while the second type just need an answer from the user, any format would work. If a small talk question of the type *yes/no* is being asked, to ensure that the user answers accordingly, two buttons will be displayed, with the intention of guiding the user into the right conversation path. An example is shown in figure 4.14, the small talk statement being "Sometimes I feel

like I don't understand my relative's needs well enough, do you know what I mean?" In the scenario of the figure, the user has answered yes, to which the chatbot has answered with another statement, also part of the small talk. Now the statement, though it is expecting an affirmative or negative answer, it does not provide the user with the buttons. This way, the conversation isn't so repetitive. Therefore, there are intents that can be trained with affirmative responses of various formats; as long as the correct context is enabled and the agent well trained, there should be no problem. This of course rises the issue of training, if the user were to answer in a very complex or unusual way, the intent would probably not be detected, that is why it is not recommended to abuse of these kind of intents.

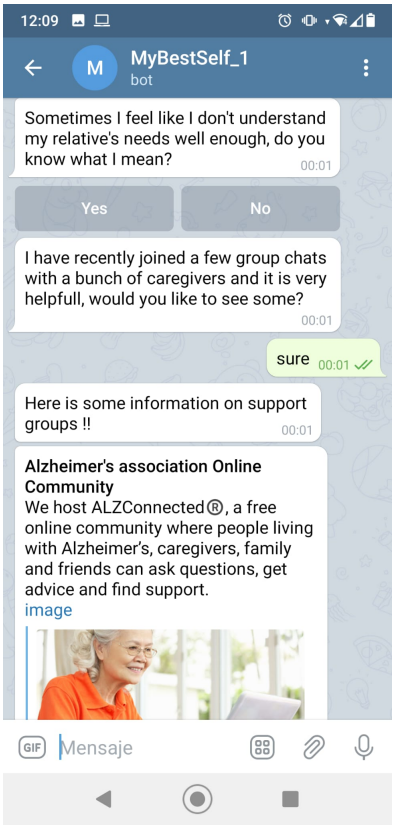


Figure 4.14: Small talk conversation. Part1.

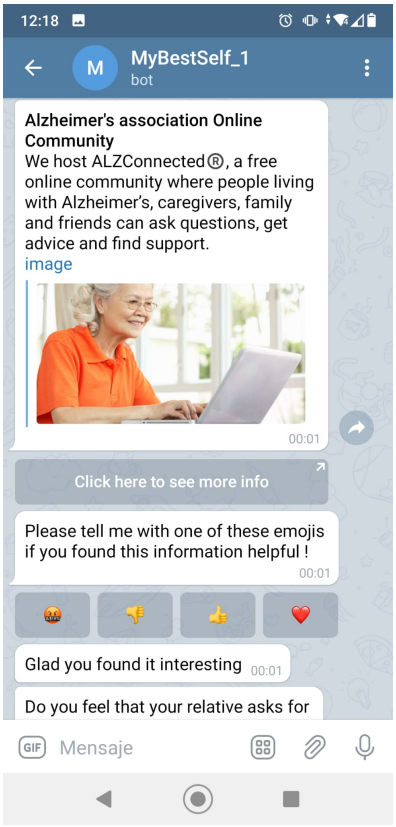


Figure 4.15: Small talk conversation. Part2.

Depending on how the conversation develops, the agent will give additional information to the user, if the user requires it. For example, in figure 4.14 and figure 4.15, the user has asked for information on support groups, and the chatbot has answered in the form of a card. A card

is an element of dialog fulfillment that was added at the very beginning of the index.js file (see figure 4.2 line 5). All cards have to show a title, a text, a picture and a button for them to display correctly. To create a card, the code executes the function of listing 4.11.

```
async function extractInfo(collectionName,collArray,final){
   let i = Math.floor(Math.random() * (collArray.length));
   let docI = collArray[i];
   const cardInfo = db.collection(collectionName).doc(docI.toString());
   const doc = await cardInfo.get();
   let info = doc.data();
   agent.add(new Card({
     title: info.name,
     imageUrl: info.imageUrl,
     text: info.text,
     buttonText: 'Click here to see more info',
     buttonUrl: info.infoUrl
   })
   );
   .
   ..
   ...
 }
```

Listing 4.11: Function used to send cards.

Since both Telegram and Dialogflow support Cards as a response format using the Card constructor does the trick; but this is not always the case. For example to send buttons, a custom payload has to be created and sent. An example is shown in listing 4.12, where the custom payload for yes/no buttons is created. The mark `inline keyboard` is used to display buttons like in the figure 4.14. There are other marks that will be explained later.

```
let pl_tl = {
  "telegram": {
    "text": text,
    "reply_markup": {
      "inline_keyboard": [
        [
          {
            "text": "Yes",
            "callback_data": "yes"
          },
          {
```

```
12          "text": "No",
13          "callback_data": "no"
14        }
15       ]
16     ]
17    }
18  }
19 };
20 agent.add(new Payload(agent.TELEGRAM, pl_tl, { sendAsMessage: true,
       rawPayload: true }));
```

Listing 4.12: Function used to send cards.

Going back to the issue of cards with relevant information; as explained earlier, depending on the flow of the conversation the chatbot will give additional information or not. For situations where for example the user is feeling uninformed, the chatbot might offer some information on support groups; if on the other hand the user is feeling stressed or anxious, the agent will send, if requested, information on relaxation techniques. In total, 6 categories of aid have been defined for this kind of scenarios (table 4.3).

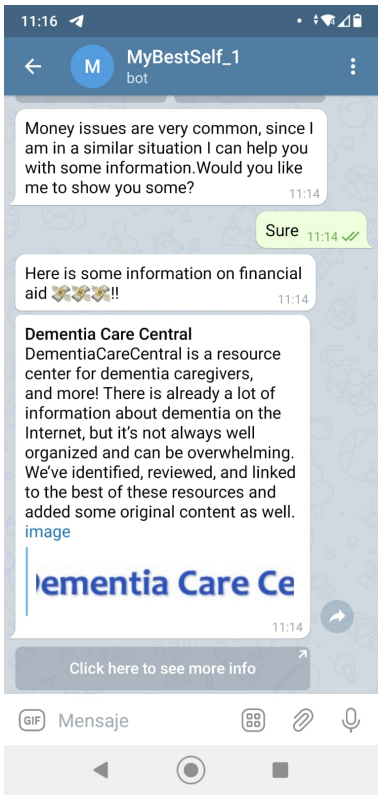
Category	Content
AidInfo	Information about economic aid available to caregivers of patients with dementia or Alzheimer's.
MBCT	Different techniques of MBCT(Mindfulness-based cognitive therapy).
Breathing	Different techniques of breathing relaxation techniques to help with anxiety.
Muscle	Different techniques of muscle relaxation techniques to help with anxiety.
Visualization	Different techniques of visualization relaxation techniques to help with anxiety.
Stories	Testimonies of real caregivers of relatives with dementia or Alzheimer's.

Table 4.3: Categories of additional information given by the system to the user.

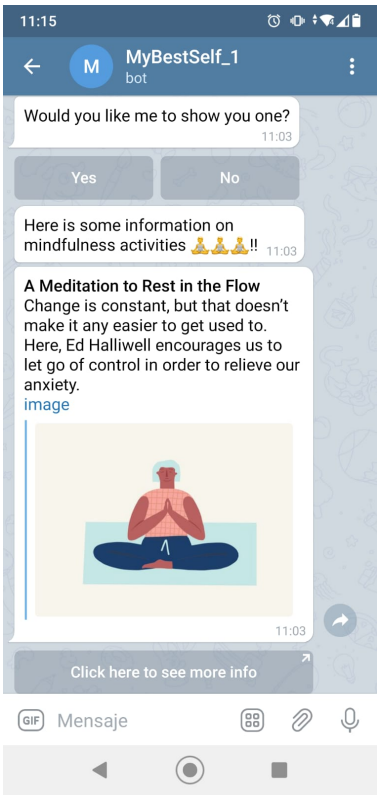
These techniques have been selected after reading various articles that back up these proceedings as effective; and the suggestions have been gathered from the most reliable sources possible. For example, the information about economic aid has been gathered from official sites like *The Bright Focus Foundation* ([24]), the *Alzheimer's association* ([23]) or the *Dementia Care Central*([45]). It is important to say that by showing this cards, the system is not appropriating the intellectual value of its content, as the cards only show an introduction and a button to go to the website where all the information is, not hiding in any way the authorship of the knowledge. Information about breathing mechanisms was extracted from Medical News Today [34], a web-based outlet for medical information and news. McMaster University [33] and Mindfulness Melbourne [8] offer some interesting examples of muscle relaxation techniques, among others([17],[44]). All the visualization techniques have been extracted from Mindful Minutes [13], a blog written by a certified instructor in Yoga and Primordial Sound Meditation. In order to add accessibility to the system, the support groups offered are online-based, like that of the *Alzheimer's association* ([23]) or even Facebook groups like the MemoryPeople Awareness Page ([37]), the Dementia Caregivers Support Group [36] or the Alzheimer's and Dementia Caregivers support chat group [39]. Feeling alone is a common symptom of caregivers [4], that is why support groups are important for the user. Another method might be to show real testimonies of caregivers around the world. The Family Caregiver Alliance [28] has a great and extensive collection of testimonies.

The need of including specific information about certain aspects of all that involves caring for a sick family member comes from the result of numerous articles, like that of Rodriguez et al. [40] who concluded that money contribution is not sufficient to cover the expenses that come with the task of caring, or that of Aguglia et al.[2], in whose study, almost all caregivers stated they were unable to support the financial burden. Other content, like for example the MBCT([10] [15]), breathing mechanisms and yoga([46] [7] [25]), muscle relaxation([35] [16] [32]) and visualization ([16] [26]) techniques were chosen due to their efficiency in reducing stress and managing difficult situations better.

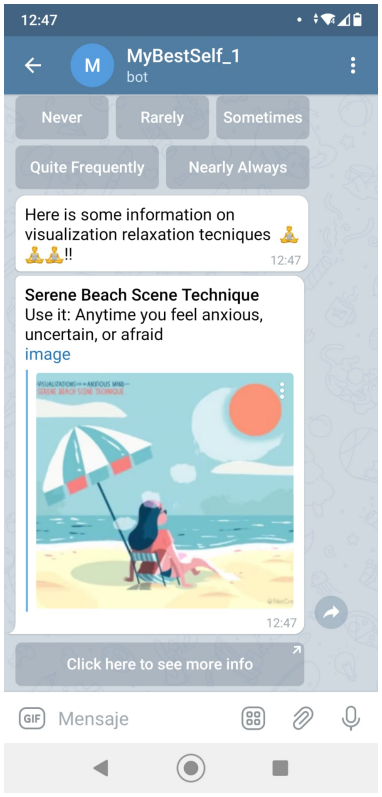
Inspirational quotes are also sent to the user depending on the conversation. Since the agent asked at the very first interaction with a user if he/she was religious or not (figure 4.13), it is able to provide quotes from whatever holly book the user follows, or non-religious inspirational quotes if the user is not affiliated with any belief.



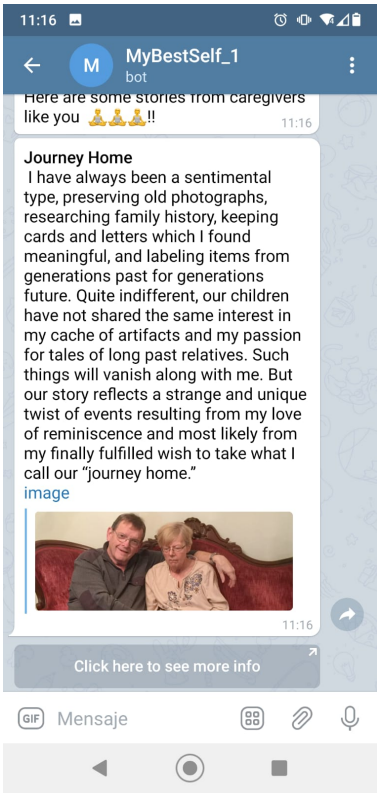
(a) Card with financial aid info.



(b) Card with MBCT techniques.



(c) Card with relaxation techniques.



(d) Card with personal stories

Figure 4.16: Examples of cards with different content.)

Sometimes, like in figure 4.15, the agent will ask the user if the information is useful, others, it will simply ask another question. This interaction does not always happen, not only because the user does not feel like seeing the information, but also because sometimes the bot will not offer it. It will instead offer inspirational quotes for example, or it will simply move on in the conversation. An example is shown in figure 4.17.

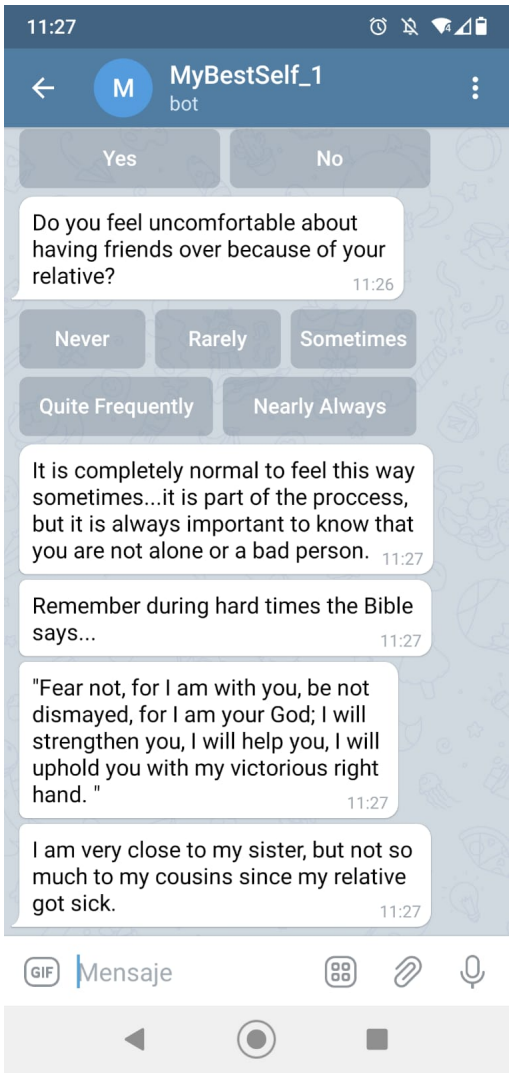


Figure 4.17: Example of religious inspirational quote.

All of these different elements work together to create a conversation that enables the system to interact with the user in a friendly way at the same time that it introduces hard questions, like in this case, Zarit

Test questions. Something very important about the Zarit Test survey is that its results have to stay within certain margins, and language is very subjective, so risking giving different answer options or interpreting language could resolve in having dubious results. That is why it has been decided that the only acceptable answers to a question from the Zarit Test are the stipulated ones, which are listed in table 4.4.

	NEVER	RARELY	SOMETIMES	QUITE FREQUENTLY	NEARLY ALWAYS
Value	0	1	2	3	4

Table 4.4: Acceptable responses to the Zarit Burden Interview

Since these questions can only receive these responses, to make the process a little bit less monotonous, 2 different display options have been configured. One is with inline buttons and the other with inline keyboard buttons. Both configurations have to be sent to Telegram through a custom Payload like that of listing 4.12. The only difference between the two is where the buttons appear in the screen. Figure 4.18 shows a Zarit Test question with normal buttons and figure 4.19 shows another Zarit Test question with inline keyboard buttons.

Figure 4.18: Zarit Test question with responses as buttons.

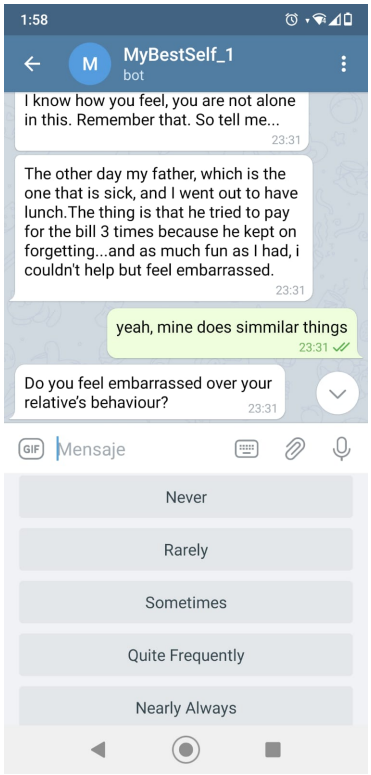


Figure 4.19: Zarit Test question with responses as inline keyboard buttons.

All of the different elements explained before interact with one another to create a conversation flow. Though the basic structure of the dialog is very clear, the way the system responses to the user might not seem very predictable, since a fairly large amount of the responses are random. This does not mean that the chatbot answers in a random manner, instead, what it does is that for a certain response from the user, the chatbot has an arsenal of possible answers that could be sent, and one is randomly chosen. The more options the better, because that way we can make the system less predictable to the user, which actually helps to disguise the psychological test. The essence of a conversation between 2 people resides on the fact that one does not know how the other is necessarily going to respond, that is why, the more options the merrier. If the system always responded the same way, the process could become predictable and would conclude with a bored user that knows how everything is going to be; making the task quite humdrum.

To summarize all this interaction a diagram has been made, figure 4.20, reflecting the actual logic of the system. When the welcome intent is detected, meaning when the user greets the chatbot, the first thing the system does is to check if the user is registered in the database. If he/she is, an available group (table 4.2) will be chosen; the criteria is that the group contains a question that has not obtained a response. When the group is randomly specified, the system will ask a small talk question and then it will randomly decide to respond to the user's answer further or directly go for a Zarit Test Question. This means that some small talk questions develop into another question, may that be to ask the user for additional information, or a follow up question, etc . The bottom line is that all small talk inquiries end up with a Zarit Test question of the chosen group. When that receives an answer, the chatbot will of course no matter what react in some way to the response, again choosing randomly within the options according to the answer of the user proceeding then to ask itself if all the questions from the Zarit Test have been completed. If so, the system evaluates the answers and acts accordingly, but if not, if there are questions from the current group still unanswered, a new small talk proceeding will start. However, if all the group is completed, a new available group has to be randomly chosen and it all starts again.

Through this process, the content of both the database and the general context change. When a small talk question is answered, its id gets removed so the chatbot does not repeat the same thing in a short period of time. When a Zarit Test question is answered, the question gets

removed from the general context's parameter *notDone* and from the notDone field of the user's document in the database. Furthermore, the answer is saved in the database's parameter *answers*. *answers* is a map, each entry consist of a key, which is the question's id, and the value of the answer. Updating these parameters is only done when an answer is received, if a user reads the question but for some reason he/she chooses not to answer, the different parameters do not get updated.

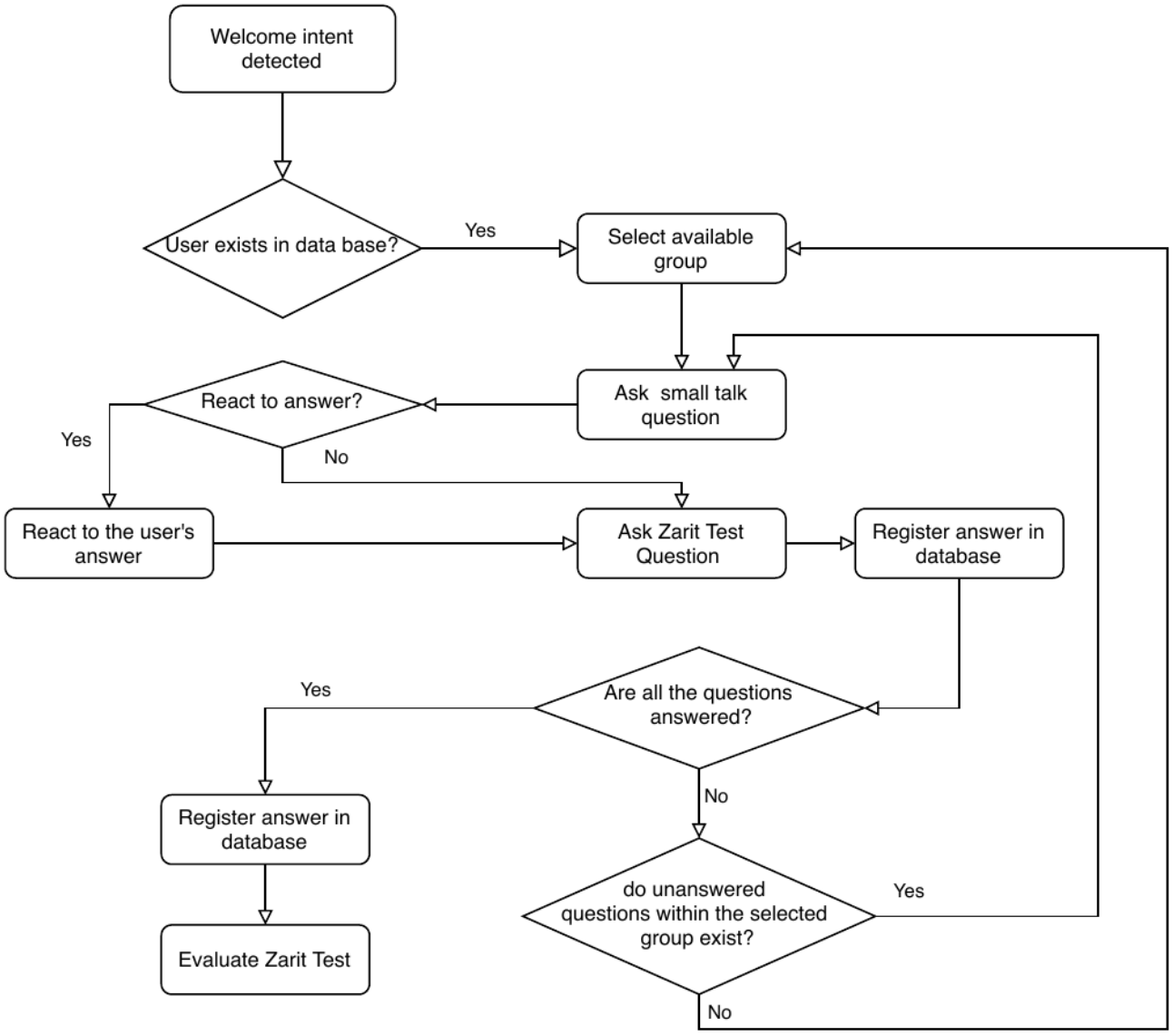


Figure 4.20: Conversation logic with a known user.

To clarify all the steps to update both the general context and the database, an simplified example in figure 4.21 has been made to showcase the different states of both the database and the general context. Keep in mind that when a group is selected, the chosen small talk question and Zarit Test question of the selected group is chosen randomly.

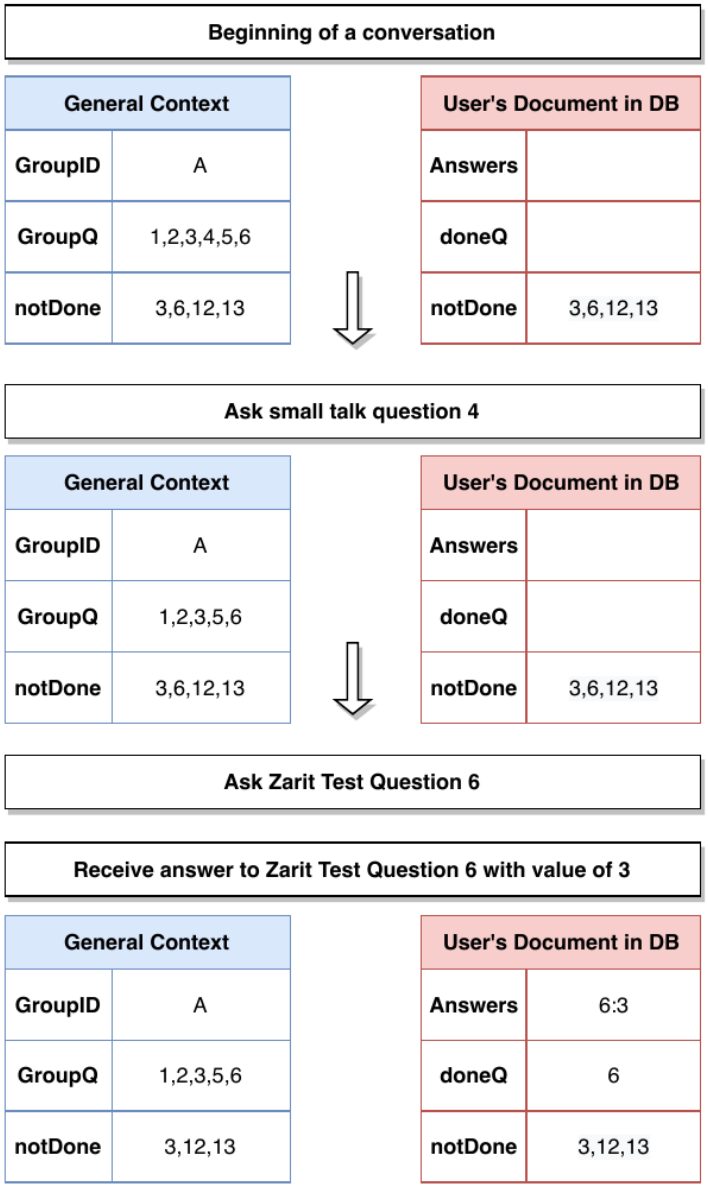


Figure 4.21: State of the general context and the user's document in the database at the different stages of a conversation.

In the diagram, the boxes of General Context display 3 sections,

the GroupID, which represents the Zarit Test Group chosen (table 4.2), the GroupQ, which are the small talk questions relevant to that group and finally the notDone field, which are the Zarit Test questions not answered. On the other box, the user's document in the database, another 3 fields were defined. All of them only represent information about Zarit Test questions only, there is nothing related to the small talk questions in a user's document. The first field is a map object that will held the answers. The second one represents the id's of the questions that have been answered, and the third the questions that have not been responded.

4.2.3 When a Zarit Test Survey gets completed

When all of the test questions have been resolved, the process of evaluating the Zarit test begins. Keep in mind that answering all the Zarit Test questions may take days sometimes, it will depend on how often the caregiver speaks with the conversational agent. The evaluation process will start only when all 22 questions have been registered in the database as answered, also understood as having an empty array in the `nodDone` field of the database. According to the Zarit Burden Interview, results can be categorize into 4 scales (table 4.5).

Interpretation of score	
0-21	Little or no burden
21-40	Mild to moderate burden
41-60	Moderate to severe burden
61-88	Severe burden

Table 4.5: Interpretation of score in The Zarit Burden Interview

The score is generated from the addition of all of the answers of the Zarit test questions, which have been stored in the parameter *answers* of the user's document in the data base. Since every time the system registers a response to one of these questions it deletes the id of such question from the parameters *notDone* of both the general context and the database. The verification of whether or not the test is complete is fairly simple. The system must check if the parameter's length is 0 or not; if it is, the test has been completed. In that case, the first thing is to reset the general context. To reset the context the function `resetContext()`, whose code is included in figure 4.13, is called. This

function resets all of the parameters of the general context except for the private information of the user.

```
function resetContext(){
    let contextVariables = agent.context.get('contextvariable');
    contextVariables.parameters.currentQ = "";
    contextVariables.parameters.currentQuestionID = "";
    contextVariables.parameters.GA = [];
    contextVariables.parameters.GB = [];
    contextVariables.parameters.GC = [];
    contextVariables.parameters.GD = [];
    contextVariables.parameters.zaritQuestionAsked = false;
    contextVariables.parameters.currentSMGroup = "";
    contextVariables.parameters.notDone = [];

    let sessionContext = {
      'name':'contextvariable',
      'lifespan': 100,
      'parameters':contextVariables.parameters
    };
    agent.context.set(sessionContext);
  }
```

Listing 4.13: Function used to send cards.

The second step is to calculate and save the score in the user's profile. To do so, the first thing is to add all of the answers, proceeding to create the actual date and time in which the test was finished. This information, together with the complete set of answers is saved in a new parameter of the user's document called `history`. `History` is composed of entries identified by the date and time (`hh:mm:ss/mm/dd/yyyy`) the fields were added to the database. Each entry contains the score of that evaluation together with the field of `answers`, which was the map object already existing in the user's document. This parameter is very useful to study the development of the burden of the caregiver taking the test, as well as seeing in what question specifically the user has improved or not.

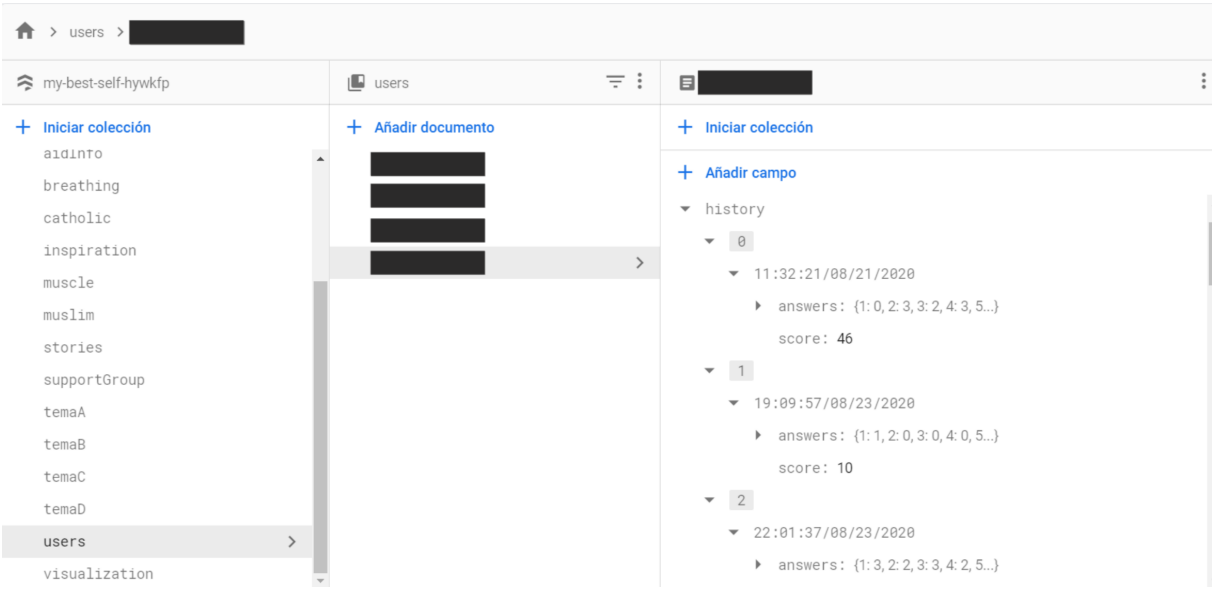


Figure 4.22: Screenshot of the parameter `history` with 3 entries.

Figure 4.22 shows an example of this parameter with 3 entries. The first entry was made the 21st of august of 2020 at 11:32 am, the score for that reading is of 46, which means the user was feeling moderate to severe burden (see table 4.5 to see the intervals of scores), and the parameter of answer is displayed, with the id of the question together with the score.

To save the data in the profile of the user, something like the code in figure 4.14 could work.

```
async function saveDateAnswers(dateOfToday,score,arr){
   let user = agent.context.get('contextvariable').parameters.userID;
   let data = {
     "score":score,
     "answers": arr
   };
   const saveHistory = db.collection('users').doc(''+user);
   const updating = await saveHistory.update({
     history: admin.Firestore.FieldValue.arrayUnion({[dateOfToday]:data})
   });
}
```

Listing 4.14: Function used to save history entries.

Note that the userID is still available in the general context, as there was no use in resetting that parameter in listing 4.13. To add a new entry

in a Firestore object the cloud function `arrayUnion` is very helpful, as it has internalized the `merge:true` feature and is more efficient than the `update` function. It also ensures that only an element that was not already present is added, which `update` or `set` don't do (See [9] for more information about `arrayUnion()`).

After successfully adding the new entry, the agent reacts to the score. There are different options within each category of burden level. The different categories of additional information are explained in table 4.6.

Opt. \ Burden	0-21	21-40	41-60	61-88
Visualization	X	X		X
Breathing	X	X		X
Muscle	X	X		X
Testimonies		X	X	X
MBCT			X	X
Support			X	X
Economic			X	X
None	X	X	X	X

Table 4.6: Options given to the user after evaluating the Zarit Test results.

Not all the time all of the options will be displayed, as always, what shows and what doesn't is a randomized process. An example of the visualization of how the user can choose an option is shown in figure 4.23.

Last, but not least, after the user has chosen an item, whatever one, the last step is to reset the database. This is safe since all the relevant information is correctly logged in the `history` parameter. This is the last step because to reset some fields, Firestore can take up sometime, and since the functions are `async` and an `await` command has to be used, the conversation could be held up a while; producing errors. Keeping this process at the end of the conversation flow ensures there is nothing waiting for the update to finish. The system, at this point says it has to leave, to ensure the conversation does not continue. The reason for which this procedure takes a little longer than usual is because Firestore actually recommends to erase the entries of array/map objects using the `arrayRemove()` function, as it is recommended as the most efficient and save. The inconvenience of this function is that it erases the fields

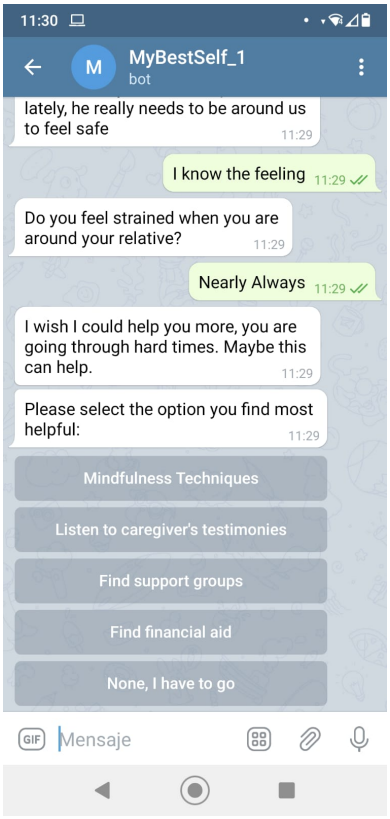


Figure 4.23: Example of religious inspirational quote.

one by one, so if the object is long, the action is long too. It has to be understood that what may seem a long or short period of time in a normal developing environment may not apply to this context. Functions here have a limited run time, and if the action takes longer than that limit, the function will crash. This has had to be taken into account when developing the project, as not a very complex logic could be built up, due to time limitations. In summary, resetting the database is left for the very last step of a conversation to ensure an uninterrupted conversation flow with the user. It is also important to mention that in case the user does not terminate the conversation, the welcome function also checks to see if all questions have been asked before proceeding. The code to reset the database is displayed in figure 4.15.

```
async function resetDatabase(){
    let user = agent.context.get('contextvariable').parameters.userID;
    let deleting;
    const updates = {};
    const resetDB = db.collection('users').doc(''+user);
    for(let i =1; i<=22;i++){
```

```
7       updates[`answers.${i.toString()}`] = admin.firestore.FieldValue.delete();
8       await resetDB.update(updates);
9       deleting = await resetDB.update({
10        doneQuestions: admin.firestore.FieldValue.arrayRemove(i)
11      });
12    }
13    const updating = await resetDB.update({
14      notDone: fullNotDone
15    });
16  }
```

Listing 4.15: Function used to reset the database.

Chapter 5

Results and discussion

5.1 Results

The intention of this project was to develop a conversational agent available to caregivers and easy to use. The bot, in theory, should be able to offer some small talk, ask psychological test questions and act upon the results received. Due to the fact that this development took place during a world pandemic, finding people to test the chatbot was difficult. Not only because of Covid-19, but also because the chatbot has been developed in English. Nonetheless, 3 people were able to use the chatbot for some time. The demographic of the users is specified in table 5.1 .

Gender	Age	Current Job	Sick family member status	Closeness to family member
Female	53	Housekeeper	father	very close
Female	24	Civil Servant	grandfather	medium
Male	55	Civil Servant	mother	very close

Table 5.1: Demographic of people that used the chatbot.

To study the usability of the system the users completed the *System Usability Scale* (SUS) [5]. A test that consists of a 10 item questionnaire with 5 possible answers each, from *Strongly agree* (five) to *strongly disagree* (one). The effectiveness of this test has been proven ([3] or [30]). SUS has become a reference in evaluating usability in the industry as it is very easy to administer to participants. It can be used on small sample sizes with reliable results and it can effectively differentiate between

usable and unusable systems.

The formula on figure 5.1 has been used to calculate the final score of the SUS test. Basically, since all odd-numbered questions are in a positive tone, the ideal situation would be to receive the maximum score, 5, in these questions. All the even questions however have a negative tone and the best score would be the lowest. Once all questions have been responded, first subtract one from each odd-numbered question and then add that amount. Next, subtract the points of each even-numbered question from five and add the total amount. Finally, add the two values together and multiply it by 2.5.

$$Score = 2.5 \times \sum_{n=1}^{10} \left(\left(\frac{2n}{n} \in \mathbb{Z} \right) \rightarrow (5 - S_n) \right) \wedge \left(\left(\frac{2n+1}{n} \in \mathbb{Z} \right) \rightarrow (S_n - 1) \right)$$

Figure 5.1: The System Usability Scale formula.

It is worth mentioning that the demographics of the users that tried the system is not representative to caregivers nowadays. According to the *Alzheimer's association ®*([23]), about 1 in 3 caregivers is 65 or older and most caregivers (66%) live with the person with dementia in the community. In our case study, only user C lived with the patient.

5.1.1 SUS values for user A

User of table 5.2 stated that she found the tool interesting, although she needed some indications as to how the system actually worked. While she found the tool needed more variety in its answers, she also said that she *"considered this tool as an opening door to a more sophisticated instrument to both comfort the caregivers and provide the community with interesting data"*. She also said that the system *"took away a portion of the load of anxiety that caring of an ill-loved implies"*. This user spent a total of a month caring for her father, who lives abroad and is in stage 2 of Alzheimer's. She said about her experience in caring that it took a toll on her and after using the chatbot is actively seeking more help for her parents.

	1	2	3	4	5
I think that I would like to use this system frequently.					X
I found the system unnecessarily complex.	X				
I thought the system was easy to use.					X
I think that I would need the support of a technical person to be able to use this system.			X		
I found the various functions in this system were well integrated.				X	
I thought there was too much inconsistency in this system.		X			
I would imagine that most people would learn to use this system very quickly.					X
I found the system very cumbersome to use.		X			
I felt very confident using the system.				X	
I needed to learn a lot of things before I could get going with this system.			X		

Table 5.2: Answers of user A to the SUS questionnaire.

Total SUS score: 80.

5.1.2 SUS values for user B

	1	2	3	4	5
I think that I would like to use this system frequently.					X
I found the system unnecessarily complex.	X				
I thought the system was easy to use.					X
I think that I would need the support of a technical person to be able to use this system.				X	
I found the various functions in this system were well integrated.				X	
I thought there was too much inconsistency in this system.		X			
I would imagine that most people would learn to use this system very quickly.					X
I found the system very cumbersome to use.		X			
I felt very confident using the system.				X	
I needed to learn a lot of things before I could get going with this system.		X			

Table 5.3: Answers of user B to the SUS questionnaire.

Total SUS score: 80.

The answers of user B were very similar to that of user A. There is a slight change when the user was asked if they found the system easy to use. She is also not as close to her relative and spent a week with her relative. She stated that she wished the conversation was more like a conversation between people, and suggested for future updates that the bot gives a more personalized feedback. She also admitted to find the system interesting and the information helpful.

5.1.3 SUS values for user C

Last but not least, user C, who lives with his relative, said the system offered some help and some useful information. He did not know for example of the existence of support groups of caregivers and said he would look more into that. He agreed with the other two users that the conversation needed more humanization, but that *"it was a starting point to develop more tools like this one"*.

	1	**2**	**3**	**4**	**5**
I think that I would like to use this system frequently.				X	
I found the system unnecessarily complex.	X				
I thought the system was easy to use.					X
I think that I would need the support of a technical person to be able to use this system.				X	
I found the various functions in this system were well integrated.				X	
I thought there was too much inconsistency in this system.		X			
I would imagine that most people would learn to use this system very quickly.					X
I found the system very cumbersome to use.		X			
I felt very confident using the system.			X		
I needed to learn a lot of things before I could get going with this system.			X		

Table 5.4: Answers of user C to the SUS questionnaire.

Total SUS score: 72.5.

The average score of the SUS test was of **77.5**, categorized as good in the usability scale [5].

5.2 Discussion

Even though the results are very similar, it is no surprise that for example user B scored higher in the questions that addressed the matter of the system being easy to use in terms of guidance and instructions. This could be due to the fact that the user is younger and is more in contact with different types of applications. User C scored the lowest in the SUS scale, the reason might be that he actually lived with the sick family member and felt the need of more personalized help; which justifies his scores.
In general the results were positive, and the feedback helpful. The fact that all three users agreed the bot needed to be more human like addresses one of the limitations of cloud-based chatbots. These bots are mainly rule-based and need a lot of interaction with the user to develop a consistent tendency. It is fair to say that the users did not complaint about the conversation being humdrum, and gave positive feedback about all the additional information given. The issue about variety in the conversation could be fixed adding more options for the bot to use when answering a user.

5.2.1 Limitations

The are a fair number of limitations that come with using cloud based platforms to develop a conversational agent. First of all, even though the development started within the free billing plan of Firebase, the billing plan had to later be updated to the Blaze plan; which consisted of a "pay as you go" philosophy. Even if this was inexpensive, it is an inconvenience. Other errors also occurred as billing had to be activated for Cloud Build API as well. This, added to the fact that the logs that referenced these issues were poorly self explanatory added a lot of lost time trying to fix an issue that had a simple solution. It is fair to say that the Firebase support team had to be contacted on 2 occasions and responded within 24 hours.

Another issue is that one’s system depends entirely on the platform’s server used to develop it. On one occasion during the development of this project, Firebase reported an error that resulted in some of its services to fall, concluding in a full day of inactivity in development or usage. Firestore also implied that the logic of functions was not very long, as a limited run time is used. In our case this limitation was not a prob-

lem, but it is true that if someone is thinking on using Dialogflow and Firestore for a project like this one should take this aspect into account.

As for Dialogflow specifically, the first restriction that comes to mind is that it has not implemented yet the feature of having the chatbot speak to the user first. There was a work-around to this issue, linking DialogFlow to Google Calendar and having Dialogflow create events on the calendar. The calendar could then send a reminder to the user that he/she has to speak to the chatbot. This however was not included within the free plan of Dialogflow and had some bad reviews from people that had implemented this solution. Using Google Calendar also implied adding more complexity for the user, as that way, the system had to obligate the user to have a Gmail account, insert his/her email and install and know a little bit of how to use Google Calendar. As one of the main objectives of the system was to make it accessible and easy to use, the option of integrating Google Calendar was discarded.

Furthermore, the project began with Node.js 8 as its version, however Dialogflow has decided to stop supporting Node.js 8 and the project had to be updated to Node.js 10. Although the code did not change, to run Node.js 10 in Firestore needs the Blaze pay-as-you-go plan to be enabled, which really gets rid of the option of developing your project for free with these tools.

Chapter 6

Conclusions and future work

6.1 Conclusions

The objective of this project was to design and implement a conversational agent that would be able to administer psychological test to users as a conversation is taking place. It would also have to asses the quality of life of the caregiver and act accordingly. The results gathered in chapter 5.1 make us conclude that the system was, in average, useful.

In terms of achievements and requirements fulfilled (see chapter 3 section 3.1), the system has accomplished to provide a psychological test, save the data for future studies, act on the results and give feedback to the user. It has also been successfully deployed to a user-friendly application (Telegram). It has, to some extent, also been able to introduce a small-talk conversation, though this point could be improved.
It was unable however to make the system start a conversation. This is an issue of Dialogflow as this feature has not yet been implemented by Google.

6.2 Future Work

This project could be seen as a starting point for the future development of better apps. Not only are we speaking of future conversational

agents, but also of the integration of these chatbots with external health care applications. This way, health care professionals could have a better understanding of the development of the caregiver burden. Medics could also, through the collected data, give more helpful feedback to caregivers. It would also be interesting to analyze better/more efficient ways to introduce small talks into the process, although probably the answer to this issue is to implement the conversational agent with artificial intelligence.

The resources could also be used to increase the data gathered about caregivers, useful information for other studies. It would also be recommended that the system is tested by more caregivers.

Bibliography

[1] Tedros Adhanom. *Worl Health Organization*. url: https://www.who.int/es. (accessed: 10.08.2020).

[2] E Aguglia et al. “Stress in the caregivers of Alzheimer’s patients: An experimental investigation in ltaly”. In: *American Journal of Alzheimer’s Disease & Other Dementias®* 19.4 (2004), pp. 248–252.

[3] Aaron Bangor, Philip T Kortum, and James T Miller. “An empirical evaluation of the system usability scale”. In: *Intl. Journal of Human–Computer Interaction* 24.6 (2008), pp. 574–594.

[4] Rose A Beeson. “Loneliness and depression in spousal caregivers of those with Alzheimer’s disease versus non-caregiving spouses”. In: *Archives of psychiatric nursing* 17.3 (2003), pp. 135–143.

[5] John Brooke. “System usability scale (SUS): a quick-and-dirty method of system evaluation user information”. In: *Reading, UK: Digital Equipment Co Ltd* 43 (1986).

[6] Gillian Cameron et al. “Best practices for designing chatbots in mental healthcare–A case study on iHelpr”. In: *Proceedings of the 32nd International BCS Human Computer Interaction Conference 32*. 2018, pp. 1–5.

[7] Yu-Fen Chen et al. “The effectiveness of diaphragmatic breathing relaxation training for reducing anxiety”. In: *Perspectives in psychiatric care* 53.4 (2017), pp. 329–336.

[8] Jenny Clifton. *Mindfulness Melbourne*. url: https://soundcloud.com/mindfulness-melbourne/progressive-muscle-relaxation. (accessed: 26.08.2020).

[9] Google Cloud-Firestore. *Function arrayUnion of Firestore*. url: https://cloud.google.com/firestore/docs/manage-data/add-data. (accessed: 27.08.2020).

[10] Rebecca N Collins and Naoko Kishita. “The effectiveness of mindfulness- and acceptance-based interventions for informal caregivers of people with dementia: a meta-analysis”. In: *The Gerontologist* 59.4 (2019), e363–e379.

[11] Claudia Cooper et al. “Coping strategies, anxiety and depression in caregivers of people with Alzheimer’s disease”. In: *International Journal of Geriatric Psychiatry: A journal of the psychiatry of late life and allied sciences* 23.9 (2008), pp. 929–936.

[12] John Correlli. *Team Gantt*. url: `https://www.teamgantt.com/`. (accessed: 10.08.2020).

[13] Melissa Eisler. *Mindful Minutes*. url: `https://mindfulminutes.com/ease-anxiety-with-visualization-techniques/`. (accessed: 26.08.2020).

[14] Danielle Elmasri and Anthony Maeder. “A conversational agent for an online mental health intervention”. In: *International Conference on Brain Informatics*. Springer. 2016, pp. 243–251.

[15] Andy Finucane and Stewart W Mercer. “An exploratory mixed methods study of the acceptability and effectiveness of mindfulness-based cognitive therapy for patients with active depression and anxiety in primary care”. In: *BMC psychiatry* 6.1 (2006), p. 14.

[16] Pauline A Fisher and HS Laschinger. “A relaxation training program to increase self-efficacy for anxiety control in Alzheimer family caregivers”. In: *Holistic nursing practice* 15.2 (2001), pp. 47–58.

[17] M.D Fredrick M Wigley. *Johns Hopkins Rheumatology - Baltimore, MD*. url: `https://www.hopkinsrheumatology.org/`. (accessed: 26.08.2020).

[18] Linda K. George and Lisa P. Gwyther. “Caregiver Weil-Being: A Multidimensional Examination of Family Caregivers of Demented Adults1”. In: *The Gerontologist* 26.3 (June 1986), pp. 253–259. issn: 0016-9013. doi: `10.1093/geront/26.3.253`. eprint: `https://academic.oup.com/gerontologist/article-pdf/26/3/253/1628590/26-3-253.pdf`. url: `https://doi.org/10.1093/geront/26.3.253`.

[19] Ma Teresa González-Salvador et al. “The stress and psychological morbidity of the Alzheimer patient caregiver”. In: *International journal of geriatric psychiatry* 14.9 (1999), pp. 701–710.

[20] Google. *Dialogflow documentation*. url: `https://cloud.google.com/dialogflow/docs`. (accessed: 10.08.2020).

[21] Google. *Dialogflow documentation. Fulfillment*. url: https://cloud.google.com/dialogflow/docs/fulfillment-overview. (accessed: 10.08.2020).

[22] Igor Grant et al. “Health consequences of Alzheimer’s caregiving transitions: effects of placement and bereavement”. In: *Psychosomatic Medicine* 64.3 (2002), pp. 477–486.

[23] Jerome H.Stone. *Alzheimer’s Association*. url: https://www.alz.org/. (accessed: 26.08.2020).

[24] Stacy Pagos Haller. *BrightFocusFoundation*. url: https://www.brightfocus.org/. (accessed: 26.08.2020).

[25] Norma E Hernandez and Sara Kolb. “Effects of relaxation on anxiety in primary caregivers of chronically ill children”. In: *Pediatric Nursing* 24.1 (1998), p. 51.

[26] Ming Shiow Huang. “Coping with performance anxiety: College piano students’ perceptions of performance anxiety and potential effectiveness of deep breathing, deep muscle relaxation, and visualization”. In: (2011).

[27] Dipesh Kadariya et al. “kBot: Knowledge-Enabled Personalized Chatbot for Asthma Self-Management”. In: *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE. 2019, pp. 138–143.

[28] Kathleen A. Kelly. *family caregiver alliance*. url: https://www.caregiver.org/. (accessed: 26.08.2020).

[29] Soomin Kim, Joonhwan Lee, and Gahgene Gweon. “Comparing data from chatbot and web surveys: Effects of platform and conversational style on survey response quality”. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 2019, pp. 1–12.

[30] James R Lewis. “The system usability scale: past, present, and future”. In: *International Journal of Human–Computer Interaction* 34.7 (2018), pp. 577–590.

[31] Jeffrey S Markowitz et al. “Health-related quality of life for caregivers of patients with Alzheimer disease”. In: *Alzheimer Disease & Associated Disorders* 17.4 (2003), pp. 209–214.

[32] Reza Masoudi et al. “Effect of progressive muscle relaxation program on self-efficacy and quality of life in caregivers of patients with multiple sclerosis”. In: *JQUMS* 15.2 (2011).

[33] William McMaster. *Student Wellness Centre. Mindfulness and Relaxation*. url: https://wellness.mcmaster.ca/topics/mindfulness-and-relaxation/. (accessed: 26.08.2020).

[34] Healthline Media. *Medical News Today*. url: https://www.medicalnewstoday.com/. (accessed: 26.08.2020).

[35] Ms Palak Patel. "A study to assess the effectiveness of progressive muscle relaxation therapy on stress among staff nurses working in selected hospitals at Vadodara City". In: *IOSR Journal of Nursing and Health Science* 3.3 (2014), pp. 34–59.

[36] Rick Phelps. *Dementia Caregivers Support Group*. url: https://www.facebook.com/groups/672984902717938. (accessed: 26.08.2020).

[37] Rick Phelps. *Memory People*. url: https://www.facebook.com/Memory-People-126017237474382/. (accessed: 26.08.2020).

[38] Clara C Pratt et al. "Burden and coping strategies of caregivers to Alzheimer's patients". In: *Family relations* (1985), pp. 27–33.

[39] public. *Alzheimer's and dementia caregivers support chat group*. url: https://www.facebook.com/groups/703409729808429/. (accessed: 26.08.2020).

[40] G Rodriguez et al. "Psychological and social aspects in management of Alzheimer's patients: an inquiry among caregivers". In: *Neurological Sciences* 24.5 (2003), pp. 329–335.

[41] Nudtaporn Rosruen and Taweesak Samanchuen. "Chatbot utilization for medical consultant system". In: *2018 3rd technology innovation management and engineering science international conference (TIMES-iCON)*. IEEE. 2018, pp. 1–5.

[42] Abubakr Siddig and Andrew Hines. "A Psychologist Chatbot Developing Experience." In: *AICS*. 2019, pp. 200–211.

[43] Jacqueline M Stolley, Kathleen C Buckwalter, and Harold G Koenig. "Prayer and religious coping for caregivers of persons with Alzheimer's disease and related disorders". In: *American Journal of Alzheimer's Disease* 14.3 (1999), pp. 181–191.

[44] Unknown2. *Therapist Aid*. url: https://www.therapistaid.com/. (accessed: 26.08.2020).

[45] Unspecified. *Dementia Care Central*. url: https://www.dementiacarecentral.com/. (accessed: 26.08.2020).

[46] Lynn C Waelde, Larry Thompson, and Dolores Gallagher-Thompson. “A pilot study of a yoga and meditation intervention for dementia caregiver stress”. In: *Journal of clinical psychology* 60.6 (2004), pp. 677–687.

[47] Haolin Wang et al. “Social media–based conversational agents for health management and interventions”. In: *Computer* 51.8 (2018), pp. 26–33.

[48] Joseph Weizenbaum. “Eliza—a computer program for the study of natural language communication between man and machine”. In: *Communications of the ACM* 26.1 (1983), pp. 23–28.

[49] Philip Yap et al. “Validity and reliability of the Zarit Burden Interview in assessing caregiving burden”. In: *Ann Acad Med Singapore* 39 (2010), pp. 758–63.