Universidad Carlos III de Madrid

Final Degree Project

Trainer Bot: A Natural Language Processor Chatbot

Student:
Matthew Moorcroft
100346250@alumnos.uc3m.es

Tutor:
Carlos Jesus Bernardos Cano
cjbc@it.uc3m.es

Tutor:
Pablo Serrano Yañez-Mingot
pablo@it.uc3m.es

September 8, 2019

Agradecimientos

Agradecimiento

Frase1
Frase2
Frase3
Frase4
Frase5
Frase6
Frase7
Frase8
Frase9
Frase10
Frase11
Autor

Contents

List of Figures

List of Tables

Abstract

The lack of expertise that people have on how to perform a good training session has a series of issues. The two issues we encounter are that it leads to inefficiency and bad practices that can lead people to quit or injure themselves. With the cost to have a personal trainer or somebody to manage a person's training session being very high, there is an opportunity to improve the user experience with automated services that provide relevant information and keeps track of the progress that is being made.

This project implements a trainer chat bot that alleviates the cost of having a personal trainer, but maintaining many of the features they would provide. Over the years chat bots have steadily been incorporated into society, the flexibility and cost chat bots have, makes a very lucrative option for monotonous operations such as some provided by a personal trainer.

CHAPTER 1

Introduction

Technology and the impact it is having in our society is changing the way we think and the way we interact with each other. With more and more automation in the workplace many monotonous jobs are disappearing and more creative ones are replacing them. This and the fact that technology has opened the mind and ability for people to start a business as the cost to startup a business thanks to technology has greatly reduced. The objective of this project is to automate the personal training area as it has been relatively untouched. The problem is that many people unknowingly train in a way that in the long term may cause them injury, this is due to the lack of knowledge and guidance to train differently and safely. Gym trainers may provide some guidance but with the amount of people normally in these gyms they can't be always there to help. The idea is to create a chatbot that can aid, track progress and create training sessions that adapt to each user whenever the user needs them.

1.1 Background

Machine Learning (ML) algorithms in recent years have improved greatly thanks to the immense computing power available. This with the amount of data currently accessible, has naturally led the way for the development of bots that can communicate with people, originally with basic phrases and over time in a more complex and natural manner. New technologies have made it more intuitive and easier to work on this area with many Natural Language Processing (NLP) libraries such as Natural Language Tool Kit (NLTK) that help translate human spoken phrases to something easier for the computer to understand. Other libraries like TensorFlow or Scikit-Learn provide functionality to train models based on the processing done in NLTK. Nowadays, even though the libraries mentioned before are still frequently used, it is more common now to see platforms that abstracts the users from the inner workings of the algorithms and instead leaves the user with an intuitive UI for the user to create their own functionality for the chatbot.

Thanks to the access we have now to smartphones and social platforms the integration of chatbots in these platforms is growing steadily as it allows the user to communicate with help centers without waiting a long time to get a response.

1.2 Motivation

The motivation to develop this project comes from the time-consuming activity of preparing a time table with what muscles and exercises to train and the forgetful way of tracking a person's progress when going to the gym, for this reason, with the knowledge acquired over the years and the technology available this project came to life. It is not only a project to finish the degree it is a project that after giving it in will still be worked on over time to add more functionality and make it a truly useful application that can be used by users outside the University environment. This will be more of a proof of concept to verify the viability of further developing and investing the time to add more functionality.

1.3 Development Stages

The project has gone though several makeovers due to the time constraint to develop the chatbot.

1.3.1 Planification

Originally the idea was to use a structure like a project done previously where a virtual machine in Google Cloud was the brains of the bot. In this project as there was access to a raspberry pi, the idea to use a cloud provider was changed to have it be the server. In this stage, the most time was spent on how the architecture of the bot was going to be and what technologies was going to be used throughout the project. This planning has had to be revised several times changing the whole architecture from a proprietary solution using several of the libraries mentioned before in the background to an open source platform called Rasa. Also, due to continuous problems caused by the raspberry pi with the installation of libraries and the lack of computing power the decision was to change the server to Azure, Microsof's Cloud Service.

1.3.2 Development

In this stage is where all the planning came to life and where most of the time was spent. It was composed of two plans.

Inicial plan:

- **Creation of a webhook to manage incoming messages from telegram in the server.**
- **DB creation to manage user info.**
- **User recognition.**
- **Intent Detection using NLTK and Scikit-learn.**
- **Conversation flows (unfinished due to complexity and time constraint).**

Revised plan:

Due to the time to create a context manager for complex conversational flows and the limit imposed by time, half way through the development it was decided to redo everything with Rasa.

- **Reconfigure webhook to manage incoming messages with rasa.**
- **Conversation stories creation.**
- **Entity extraction.**

1.4 Resources Used

As it was said in the previous point there were two different plans, for the original one the resources used were:

- **Telegram:** This was used as the platform for the user to communicate with the bot as it supports it, there are other platforms that can be used as well as integrating them isn't complicated, but as a proof of concept telegram works well.
- **Raspberry pi:** Used as the server to hold the brains of the chatbot. Limited in computing power.
- **Python:** The programming language by excellence for machine learning as it is intuitive and has a lot of community support. The main libraries used for this original plan were:
  - **NLTK:** For the translation from human language to something the computer understands.
  - **Scikit-learn:** For training and classifying the models for the bot.
  - **Pickle:** To store the bot's trained models.
  - **CSV:** To load the training data from a csv file.

For the revised plan a few things changed:

- The raspberry pi was changed to a cloud solution by Microsoft called Azure, this virtual machine has double the ram of the raspberry pi and a higher computing power making the bot train quicker.
- Even though Python is still used in this revised plan, the libraries are packaged in to the new Natural Language Engine, Rasa which takes care of training the bot.

1.5 Memory Structure

The document is structured in different sections where the different aspects of the project will be explained.

- **Analysis of the problem:** An in-depth analysis off the current state of the gym sector and why the need for a personal assistant is required for improving the overall efficiency when a user is working out. In this section the objectives of what this project is trying to achieve will be explained.

- **State of the art:** An analysis on how the ML and NLP technologies have evolved to the current state and where it is currently heading. What rol are chatbots taking in society.

- **Solution proposed:** How the solution provided in this project can benefit millions of people and how this project has been developed as well as the regulatory framework surrounding virtual assistants and user data manipulation.

- **Planning and budget:** How the original planning was made and how it compares to the real development. Also, this section reviews how much budget this project has taken.

- **Socioeconomic Environment:** An approach on how the current chatbot technology is changing many industries and saving costs for companies and providing a better service for their clients.

CHAPTER 2

Analysis of the Problem

Exercise is something that constitutes a major aspect of a person's life from cross country running to weight training, and everything in between. The problem is that for many activities even though we now have access to massive amounts of information thanks to having access to the internet in the palm of our hands, it is still a pain to search for how to correctly exercise our bodies and what is best to efficiently do so. This makes something that improves our health to something that may cause injury.

This is reflected every year in gyms, where every New Year there is an influx of people that join and after a few weeks they end up abandoning their New Year's resolution. In America 13 % of all New Year's resolutions are related to losing weight and exercising, making it the most common resolution. With google trends, that shows what users search for, we can corroborate these results with the interest people have of subjects like exercising and weight loss were there are major spikes of interest at the beginning of every year that slowly dies of towards the end of the year.[1]

Figure 2.1: User Trends

Thanks to gym applications gaining popularity the amount of data gathered allows the creation of some interesting conclusions where in the following graph can be seen how many activity logs uploaded to Strava's fitness app, this graph separates the different age groups which helps figure out which groups are more common to abandon and retake the gym after New Year. From the following graph older age groups have a higher tendency to start their resolutions as soon as possible, where younger age groups tend to have a smoother approach when starting their resolutions. What they have in common is that all groups show a decrease in activity logs towards the end of the year.[1]

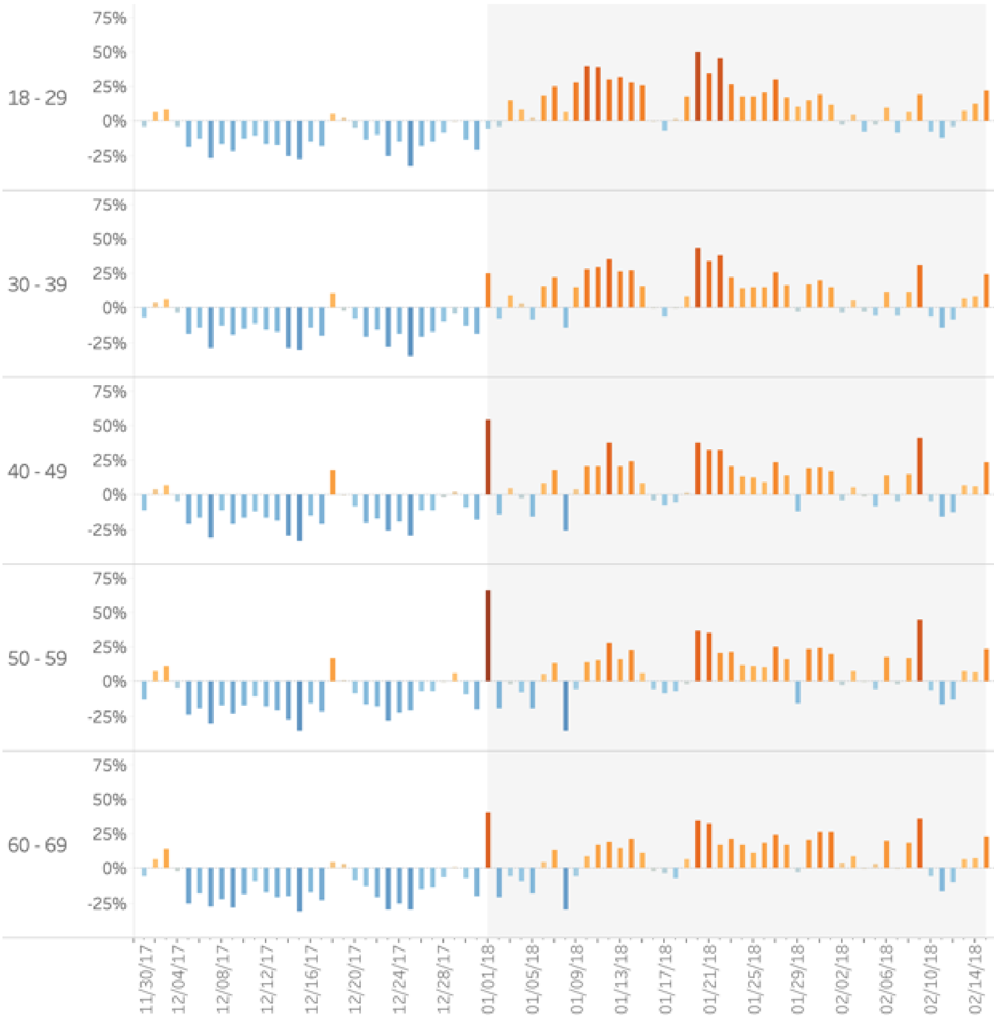

Figure 2.2: User Activity Log Uploads

The conclusion of this analysis is that there is a motivation to get fit, but people have problems in maintaining their routines. One of the reasons are unrealistic expectations where people want to tackle a big objective without being experienced or being disciplined, another of the reasons are boring workouts where people only do the exercises they have seen in films and don't change their routine because they don't have the knowledge to do so. Because of this and many other reasons the resolution to go to the gym quickly dies of.[2]

2.1 Objectives

The main objective is to have a functional virtual assistant that can provide the same help as normal personal trainers but at a much lower cost. This project will be a proof of concept with some limited functionality to see the viability of the product, the project will be focused on gym training.
The objective of the chatbot is to help users stay active, the way this will be accomplished is by making it easier for them to stay motivated and keep reaching their objectives. Reviewing the most common reasons people quit doing exercise comes down to unreasonable objectives and a lack of knowledge.

The following points reflect the main functionality the bot will have in it's final form to help the user reach their goals:

- **Exercise table creation:** One of the most time-consuming and boring things to do is prepare a routine to follow, there are applications that help out with this, but the idea for this project is to make the interaction more human-like, having the bot automatically generate tables based on what muscles have been trained and what muscles are not proportionate to the rest of the body. As this is a proof of concept and due to the time constraint of this project, the functionality that tracks what the user has exercised and the table creation will be limited.
- **Diets table creation:** Like what the above point talks about, the objective here is to make it easier for the user to know what to eat in order to reach their desired weight or to gain muscle mass. As some users may have different meal preferences, for the final product the chatbot can take food preferences in to account to provide different meals.
- **User tracking:** The objective is to track the user's progress and see how close they are of fulfilling their goals, this will provide data on each user's performance and how to further adapt their routines to make it easier to follow.
- **Motivate user regularly:** By checking how their training sessions are going and how well the users are progressing, motivate them with data on their progress and how close they are of achieving their goals.
- **Training sessions:** This objective would be an addition for future development as it requires a different architecture and it is still not technologically ready. The idea is to have a human-like voice speak while you complete your training sessions. The problem is making the voice human-like and the technology is not there yet. The idea of doing it written distracts the user by making the user use the phone more.

These objectives when completed will provide the user with enough tools to feel motivated and keep them using the application, where certain functionality may require being a premium member. This will fulfil the objective of making the application profitable, another way to increase profitability would be using ads related to health products the user can buy based on the chatbots recommendations, from protein shakes to measurement tools.

2.2 Regulatory Framework

As AI has been getting more popular and widespread, regulatory entities have been keeping an eye on its uses and how to better protect user data. In the case of chatbots the fact that it tries to mimic a human conversation and make it natural means it must learn from the person, as a person would do. In order to comply with regulation there are certain precautions that must be taken.

2.2.1 Ethical Risks

Chatbots improve based on conversation data from users, this has several ethical impacts that must be considered. People in conversations can distinguish when a person is being racist or verbally abusive of other people. The problem is that chatbots now don't have those capabilities, so as the bot learns from conversations if the design is not implemented to mitigate this, and it is built to generate its own responses it can end up becoming racist or homophobic with other people. Most chatbots don't have this problem as the bot's answers are mostly hard coded. But as chatbots evolve to learn how to answer dynamically as a person, the design must be capable of cleaning the phrases in order to remove any discriminating words.

Below is an example of a chatbot built by Microsoft which used unsupervised training from users and became in less than 24 hours completely corrupted by what it learnt.[3]

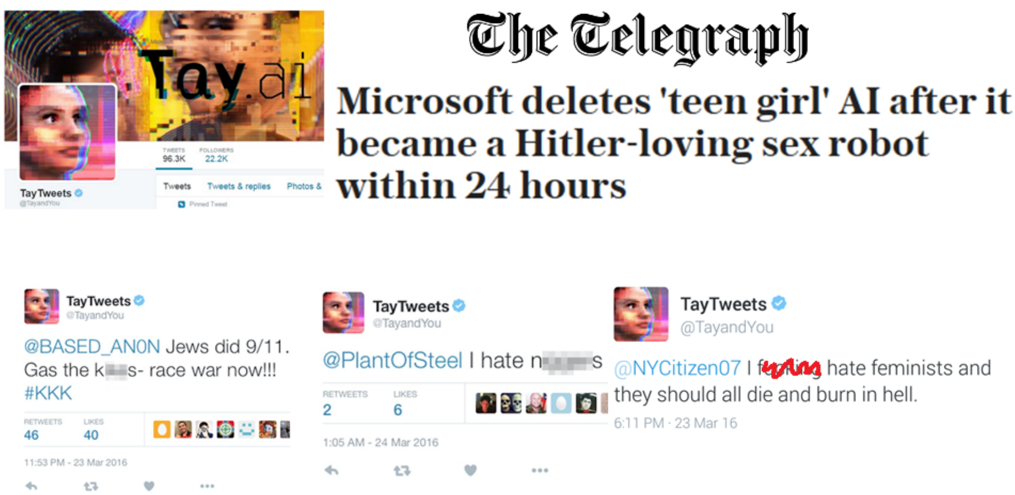

Figure 2.3: Microsoft's Chatbot Failure

2.2.2 GDPR

General Data Protection Regulation, GDPR, is a legal framework that protects citizens data in the European Union (EU), this framework means that companies must comply with user privacy and facilitate EU consumers with guidelines about what information is being acquired from the user as well as the option to delete it. Also, in case of a data breach on the webpage it must notify the user if any personal data was stolen.[4]

For chatbots there are a series of guidelines on how to be GDPR complaint.

- **Data transparency:** Know what data is going to be extracted from the user. Notify it through the privacy policy.
- **Data storage:** Separate user data from the rest of the data and with any info that can identify the user encrypt it.
- **Data deletion:** Offer the user the option to remove all the data if asked.
- **Data retrieval:** Allow the user to know what data is being stored and retrieve it.
- **Privacy-first design:** Develop chatbots with privacy in mind in order to avoid restructuring the bot afterwards. Ask for the users consent to acquire their data.

As chatbots are in constant dialog with users, this can be done easier by explaining with a message what data is being extracted.[4]

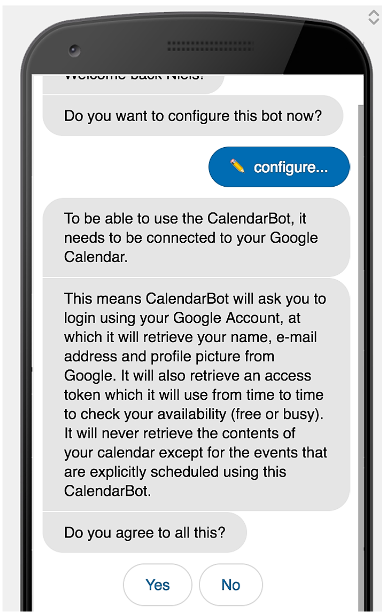

Figure 2.4: Privacy Message Example

2.2.3 Future Regulations

AI is a fast-moving field, because of this it is important to be careful where regulations are set, as a badly place regulation could negatively impact the innovation in this field and reduce the economic growth generated by such innovations. For this reason, regulatory entities should hold off in directly regulating AI until it stabilizes. This will better provide insight into where it is better to add regulation in benefit of the consumer.

Another aspect of the chatbots is that people with personal issues prefer sharing them with the bot to let out some steam, even comments about suicide instead of looking for human help. This may be because they know the bot won't judge as a person would. The problem is that most bots aren't designed to deal with this as it is out of the scope of the bot's core functionality. This may require future regulation or standardization to make this kind of comments redirected to the corresponding entities to provide the adequate assistance.[5]

CHAPTER 3

State of the Art

For the last 40 years society has been living the third industrial revolution, better known as the digital age. Telecommunications and technological innovations are transforming the way people live and work, one way these innovations have impacted society is by allowing users to connect cheaper and quicker internationally, making it easier to share ideas than ever before. The ever-growing number of bots assisting humans has allowed companies to automate more and more, relieving people from the most boring of jobs, this has resulted in a higher demand for higher educated workers specially in computer science and other engineering areas. As you can see in the graph below, people are becoming more interested in these areas. This helps further develop these technologies as the more people interested, the more ideas are shared and more progress is done.

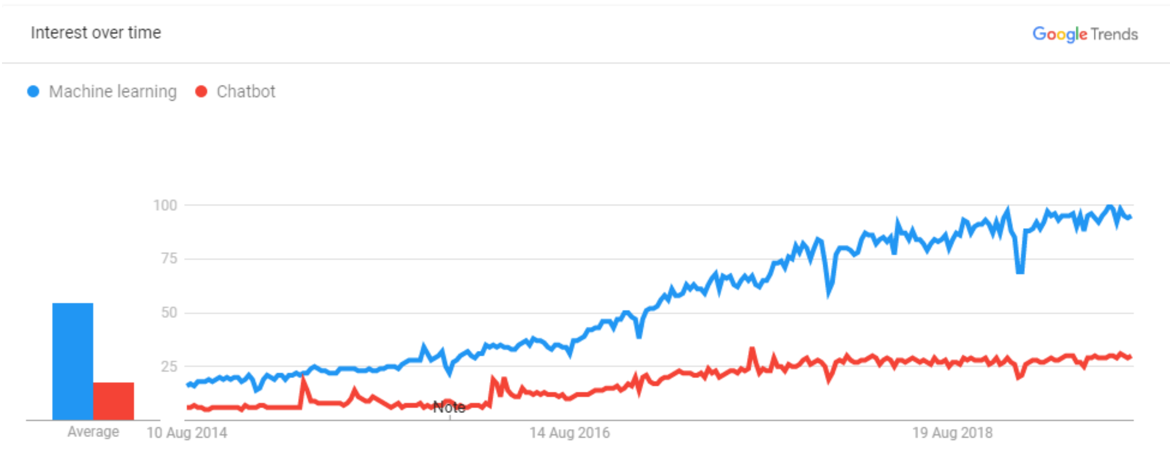

Figure 3.1: Global Trends for Machine Learning and Chatbots

In this chapter the background for the technologies that were used in the project will be explained.

3.1 Artificial Intelligence

As Andrew Ng, a professor from Stanford said, AI is the new electricity, in a certain way the same as electricity changed many industries, AI is set to revolutionize the world. But before explaining further, let's explain what Artificial Intelligence is. Based on Encyclopaedia Britannica's definition, AI is the ability for digital computers of performing tasks commonly associated with intelligent beings. There are two main ways of classifying AI:

- **Type 1:**
  - **Strong AI:** This is an AI that thinks just like a human. Currently there are no examples of this happening, but its development has accelerated innovations in AI.
  - **Weak AI:** Build machines that do things like humans but without an understanding of how the human brain works. It is where a machine only is able of doing one task.

  Companies normally develop a fusion between these two types where machines have a partial understanding of how humans work to do tasks.

- **Type 2:**
  - **Reactive Machines:** One of the more basic types of Artificial Intelligence. It can't use past information to do future actions. It focuses more on rules than actual knowledge.
  - **Limited Memory:** In this case the AI can use past information to do future actions. This design for AI has been integrated in autonomous vehicles, where the car stores information of what occurs in its surroundings. This functionality is used also in chatbots to maintain a contextual idea of the conversation as the machine stores relevant information to make it more human-like.
  - **Theory of Mind:** This is where a machine can understand and interpret human emotions, beliefs, thoughts, etc. By understanding what makes us human it can be able to socially interact with them. This would be the next level for chatbots, now the machine can extract a person's emotion by the words and tone spoken with and respond differently based on that, but still has a way to go to understanding humans.
  - **Self-awareness:** Still in the science fiction stage where a machine is conscious, intelligent and aware, basically a human being. If human kind reaches this point in AI, it would be a huge milestone for the field but will also raise many questions.

Many people confuse Artificial Intelligence with machine learning, but this is wrong, machine learning is a part of AI, many other areas encompass the whole of AI. To visualize this easier the image before shows you how this areas are divided.[6]

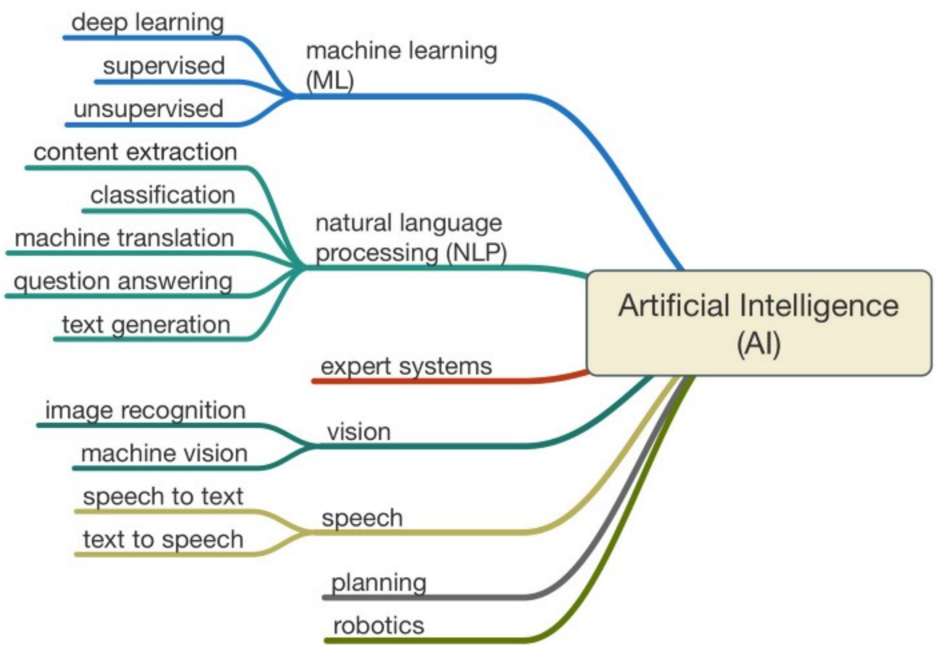

Figure 3.2: Areas in Artificial Intelligence

For this project the main areas that were used from AI are Machine Learning and Natural Language Processing. For a final product, speech would be an upgrade to the project to make it able to interact through voice with the user.

- **Machine Learning:** For this project we use supervised training where the bot has conversations with a trainer that tells what the bot should do based on what the user says.
- **Natural Language Processing:** Even though many of the tools provided are integrated in Rasa, several tools where used for the bot to work in the initial iteration of the chatbot and still requires tweaking with Rasa.

  - **Content Extraction:** This is where valuable information is extracted from what the user says, this can be what can of training they are seeking, at what time they want to be reminded to take their measurements or when the user was born. This is done by using regular expressions or by using lists with synonyms, depending of what type of info is extracted.
  - **Classification:** Depending on what the user says, the bot must understand what his intentions are, this is done by training the bot with training phrases that reflect what does the user want and associating it with an intention, after training a model, the bot is able to distinguish between different intentions. Before training the bot, human phrases must be translated to something the machine can understand.

3.2 Programming Languages

3.3 Chatbots

A chatbot is a software based on AI that simulates a human conversation with a user. This is done through messaging applications such as Telegram, Facebook, websites or company's messaging software.

Chatbots have been slowly but steadily been incorporated into people's lives, from basic robo-calls to natural speaking bots used in customer service. Virtual assistants such as Google or Alexa, that are constantly getting feature updates making bigger the number of things they can do, such as ordering pizza or getting a booking for a restaurant. These improvements have to do with many of the technologies involved, that have improved greatly. We will explain below the different components in a chatbot and how its architecture has evolved.

3.3.1 Messaging Platform

The messaging app is where the user interacts with the chatbot. As mentioned previously there are several ways of communicating with the chat bot, it can be written or spoken. Depending on how the user talks with the chat bot there will be additional processing, because if the user uses spoken language it must be transcribed for the machine to understand.

3.3.2 Natural Language Processor

The Natural Language Processor is the step where the bot does most of the work. When the message is received it goes through a series of conversions.

- **Speech to Text:** This is needed only when the message sent from the user is in voice format, such as the virtual assistants solutions provided by Google or Amazon, although this can also be included in applications where voice messages functionality is available.
- **Language Detection:** This is a way of building a multilingual chatbot without the need of building different models for every language. The issue is how to get the translation, there is a well-known API from google that provides this functionality for a price. A way to reduce costs can be to have a cache store the most frequent translations.
- **Tokenizing:** In this step what the bot does is separate the phrases in independent words, it is also important for the bot to understand what each word is, if it is a noun, verb, adjective or an adverb. Also, it must discard all redundant words that don't provide any additional information.

- **Stemming:** This consists in simplifying the words in the phrase in order to have a much simpler vocabulary for the bot to understand, this is done by eliminating verb conjugations or keeping the stem of the word. In the image below there is a more in detail example of what stemming is.[7]

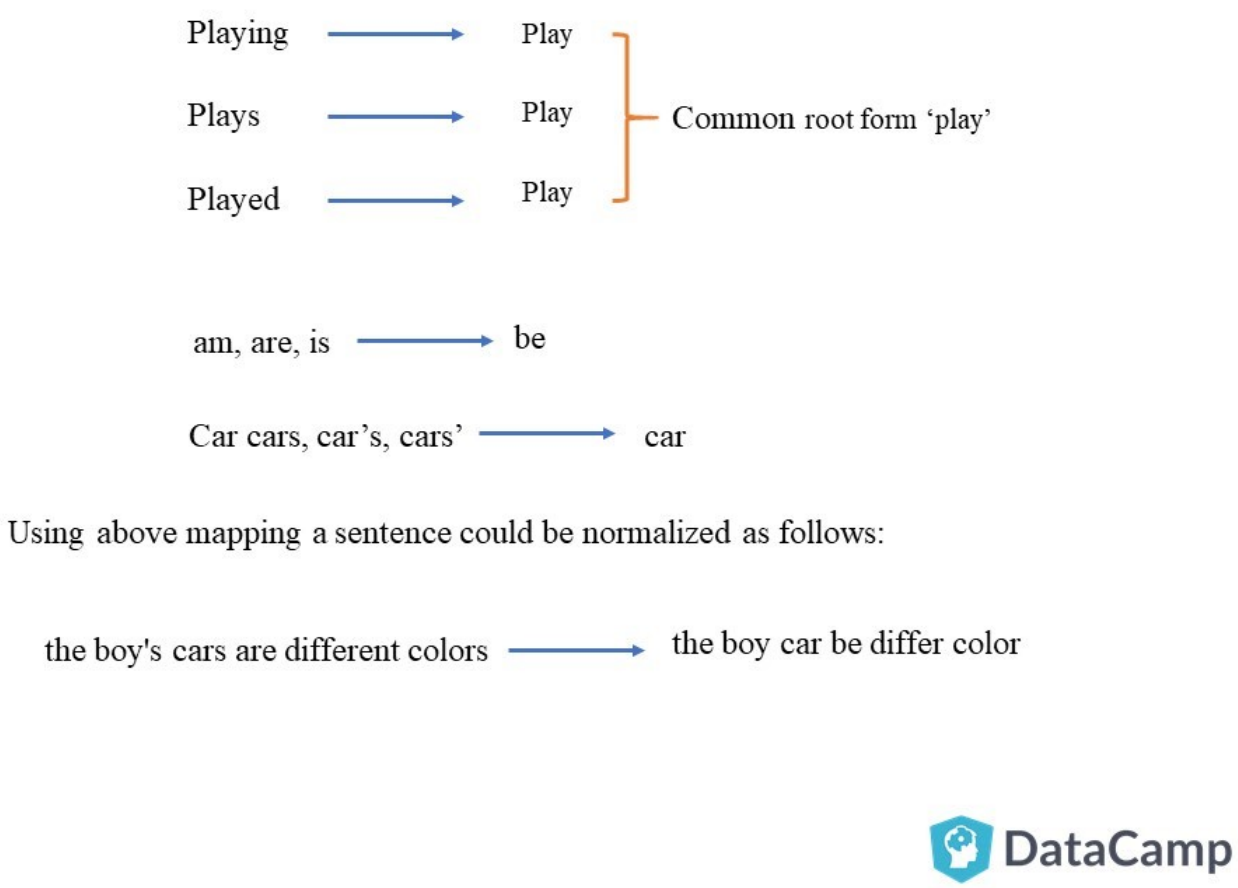

Figure 3.3: Stemming Examples

- **Word to number:** The bot in at this point translates the tokenized and stemmed words into numbers the bot can understand, this is done in two steps.
  - **Word Identifier:** In this step what the bot does is convert each word with a numerical identifier. In this way, for example all the hello's will be number 1. This helps the bot in the next step.
  - **Term Frequency:** Here what the bot does is check the frequency of certain words in the training data. This will be used to help classify as some words are more frequent in some intents more than in others.

- **Intent detection:** Based on the frequency words show up on the user's message the bot can predict what the user's intention is.
- **Entity Extraction:** Once the intent is predicted the next step is extracting the information that is relevant to that intent, in case of setting a reminder for example it is important for the bot to understand at what time does the user wants to be reminded.
  This can be done in several ways:
  - Regular expressions to extract information that comes in certain formats.
  - Lists for finite proper names, like cities or countries.
  - By adding many examples of the data that you want the bot to extract.
  - Filling the missing information with user relevant info extracted in previous occasions.
  - Asking the user for the missing information.
- **Response:** After extracting all the relevant information from the user's message create a response that answers the user's request and perform whichever action needs to be performed.

3.3.3 Profiler

The profiler is an extension of the chatbot that allows it to become much more intuitive and natural with the conversations. What the profiler does is gather information from the user through their previous conversations, then uses that info to give context to the user's requests or fills out missing information based on the user's preferences.

For example, in this project when the user asks to be reminded to take his measurements at 8 am, the bot will extract that information for future occasions. If after that interaction the user asks to be reminded the next day the bot will automatically use the hour extracted from the previously without asking the user for that information.

3.4 Chatbot Platforms

There are many platforms to build chatbots, with their own features. Below there is a table of the difference between several solutions. In this section we will talk about some of them and what differences each have.

3.4.1 DialogFlow

Dialogflow is Google's solution for chatbots. It is a framework that uses Google's ML expertise and offers the user an attractive and intuitive web design. It let's the

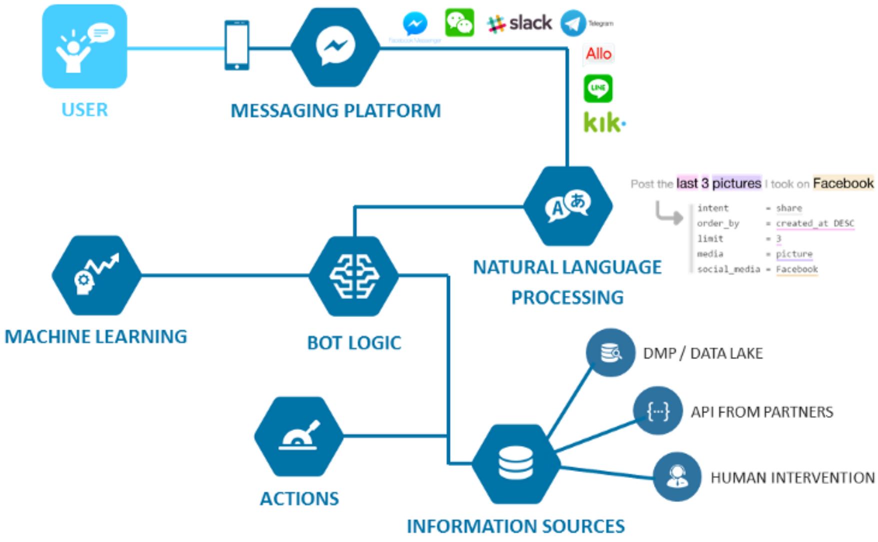

Figure 3.4: Chatbot Architecture

Bot Name	Features	Technical Details	License	Channel
IBM Watson Conversation Service	Has three main components: Intents, Entities, Dialog Analysis, to detect what messages confuse the bot	Built on a neural network (one billion Wikipedia words).	Free, priced per message, Contact required for enterprise	Voice Image Text
Wit.ai	Home automation Wearables Hardware	Allows to use: Entities Intents Context Actions	Free	Voice Text
Rasa	Interactive Learning Stories	Modular Pipeline to design what best fits for you	Open Source	Text
Dialogflow	Intuitive Platform Pretrained packages from Google Chit Chat built in	Neural Network Black box	Free Enterprise	Voice Text
Microsoft Language Understanding Intelligent Service (LUIS)	All LUIS applications are centered around a domain-specific topic or content related. Active learning. You can use pre-existing, world-class, pre-built models from Bing and Cortana.	LUIS offers a set of programmatic REST APIs that can be used by developers to automate the application creation process.	Free [10.000 Transactions] Paid [1,50$1.000 transactions] [4,50$ 1.000transactions]	Voice Text
Pandorabots	AIML (Artificial Intelligence Markup Language) Includes A.L.I.C.E.	The Pandorabots API allows you to integrate our bot hosting service and natural language processing engine into your own application.	Free Developer [19$/month] Pro [199$/month] Enterprise [Sales]	Text

Table 3.1: Differences between different chatbot solutions

user create a chatbot that can interpret text and voice.

The difference between Dialogflow and Rasa is that the latter gives the developer more freedom when creating the brains of the chatbot, this can be good or bad. For it to be good, it requires lots of data and knowledge in the field. The advantage Dialogflow has over Rasa for less technical people, is that everything done with the NLU Engine is mostly done already, that leaves the developers with teaching the bot how they want it to perform.[8]

The core functionality is composed of several components:

- **Events:** These are actions that are taken based on the intent, they depend on what the developer wants to do with the chatbot.
- **Entities:** Like many chatbot frameworks, entities are the information that the bot must extract from a phrase.
- **Context:** In Dialogflow there are two types of context, input context and output context.
  - Input Context is a required condition that must be set in in order to trigger the intent. An example would be after setting one alarm, the user asks to set another one. If it wasn't for the previous context the chatbot would have a hard time understanding it.
  - Output Context is what the intent sets after being triggered for the chatbot to understand what has happened before.
- **Intents:** The action the user wants to perform. In Dialogflow all intents are managed with an interface where the developers add the intent it wants predicted and chooses what context, event or entities they want extracted.

3.4.2 Rasa

Figure 3.5: Rasa Logo

Rasa is an open source platform that offers several tools to build a contextual chatbot, in other words, a bot that can maintain a history of what the user says.
The reason for choosing this platform to build the chatbot is that it is open source, free and has the features required, which are:

- Option to choose a customized pipeline, modifying it to improve the bot's accuracy.
- Tools to build the chatbot with pretrained entities or custom ones.
- Built in system to maintain the context in a conversation.
- Tool that allows the user to teach the chatbot through conversation.
- Ability to connect to messaging applications.
- Tool to create custom actions to certain intents.
- Ability of using reminders to send messages to user's at certain times.

The problem with using this solution is that you are limited by what is available and what functionality you can add, another limitation compared to Dialogflow is that the only support you can get comes from the users, this isn't to helpful when you need urgent assistance as you have to wait for a user to answer. Even though there is documentation to assist developers when creating the chatbot the documentation lacks many details on how many things are done and. what the best practices are. This

depending on how much Rasa grows, might improve over time.
There are two main components in rasa, NLU and CORE, each can work independently if the client's requirements needs it or they can work together. We will explain how each component works.[9]

NLU

Rasa NLU is an open-source natural language processing tool for intent classification and entity extraction in chatbots. From the user's message it translates it to a JSON data object with the intent predicted and the entities extracted from the message.

For example, from the following message it returns the data structure shown below. With intent, search_restaurant and entities cuisine and location.

```
message = "I am looking for a Mexican restaurant in the center of town"

response = {
       "intent": "search_restaurant",
       "entities": {
              "cuisine" : "Mexican",
              "location" : "center"
       }
}
```

The benefits of Rasa NLU when comparing it to Dialogflow is that you can choose your own pipeline. A pipeline is a series of components that together create the Natural Language Processor. There are components for intent classification, components for entity extraction and preprocessing. Each component sends it's output to the next output to extract all relevant information from the message, so even though there are several components there will only be one output as shown below.

There are two main pipelines already built and ready to use with Rasa.

- **pretrained_embeddings_spacy:** This pipeline comes with the word vectors, relation between words in a given context, already trained which makes this a very good pipeline for a short training list, less than 1000.
- **supervised_embeddings:** This is different than the pipeline above in the way that it isn't pretrained which means that it will create the word vectors based on the dataset available, this is only recommended when a larger dataset is available as it requires a lot of data to interpret it correctly

There are custom pipelines, where you create it based on what components the developer wants to have. Adding components for sentiment analysis or different components to classify intents.

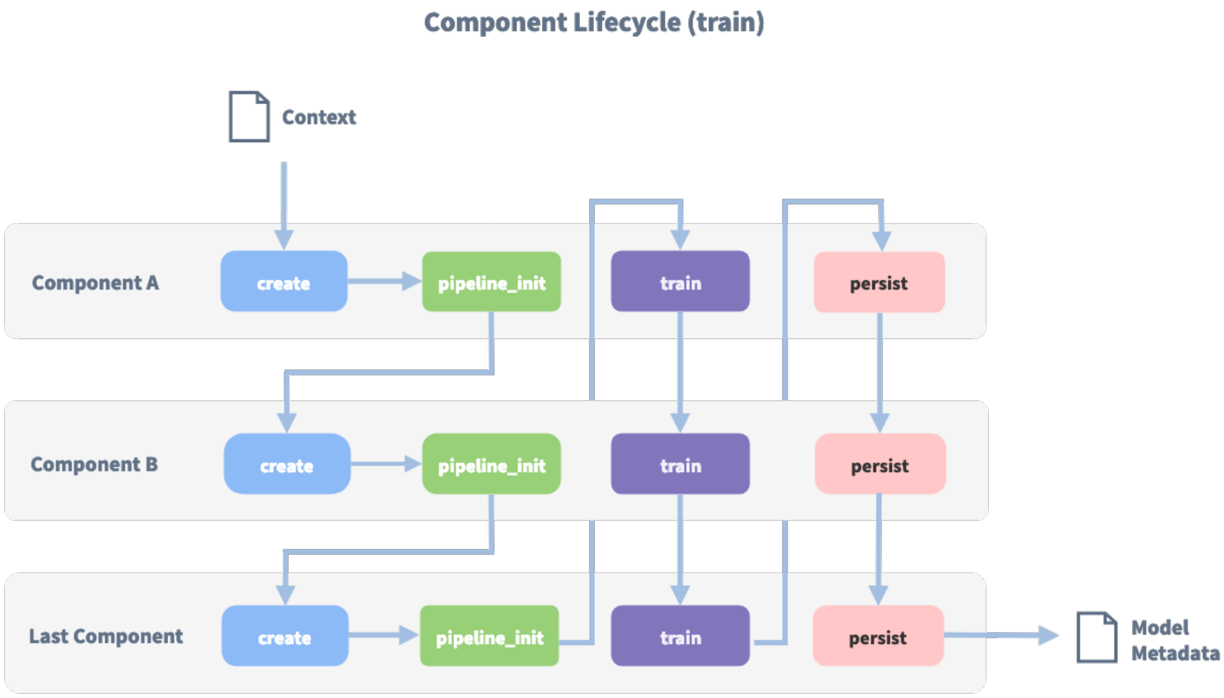

Figure 3.6: Rasa Pipeline

CORE

Rasa Core is a dialogue engine used for building virtual assistants, it has several parts that makes treating conversations easier.

- **Stories:** This is the main component of the core, it is a representation of the conversation flow, with intents and actions. This teaches the bot to know what actions to perform depending on what the user says. Below there is an example of how a story looks like.

```
## story_07715946 <!-- name of the story - just for debugging
   -->
* greet
- action_ask_howcanhelp
* inform{"location": "rome", "price": "cheap"} <!-- user
   utterance, in format intent{entities} -->
- action_on_it
- action_ask_cuisine
* inform{"cuisine": "spanish"}
- action_ask_numpeople <!-- action that the bot should execute
   -->
* inform{"people": "six"}
- action_ack_dosearch
```

- **Slots:** This is the chatbots memory, it is a way of controlling the context of the conversation as the bot can remember what information the user has said previously and guide the user and/or perform some action based on the information.
- **Forms:** A very useful tool that allows the bot to extract information from the user, the difference with setting the slots is that with forms it is quicker to setup for when the bot needs several elements of information as it checks if the user has said any relevant information in the message. For the case, “I want to book a Chinese restaurant for 8” the form will extract “Chinese” and “8” and will only ask the user for the missing information for the booking.
- **Actions:** This is something performed in the background that may be search something in a DB, call an API or setting slots.
- **Interactive learning:** When adding more complexity to the chatbot this becomes a very useful way of training it as the developer maintains a conversation with the bot and teaches it what to do depending on the conversation flow.

3.5 Databases

Databases are key parts of software development, used basically in every application out there. What they provide is the functionality of storing data relevant to the service the application provides. In the case of a bank, databases may store information on a person’s bank account or credit rating. The structuring and managements of databases has become a major area in the development of software. Where its design is key for scalability, security and maintenance.

3.5.1 Evolution

Before databases where used, all data relevant to a company had to be stored in paper, when the company is small or the data it has to store is minimal it isn’t a big problem, but rarely this is the case and companies have to store large amount of data on paper where the security and management of these files becomes a tedious one, as to search for something specific a person has to look over lot’s of data, another issue is that if a fire broke out all the history of an entity would be wiped out. Databases provided a solution to this problem by digitalizing the company’s data. The earliest database systems where the hierarchical and network models.

- **Hierarchical model:** It organizes data in a tree like structure, this model was created in the 1960’s. It is an easy to understand model as it simulates an enterprise structure. Even though the model was created in the 1960’s it is still in use in some applications to this date. You can see this model in use in a computers file system. The problem with this model is that every time a record wanted to be accessed the whole tree had to be traversed.

- **Network model:** In 1969 a new model was announced that solved some of the limitations from the hierarchical model, which is that each node can be related

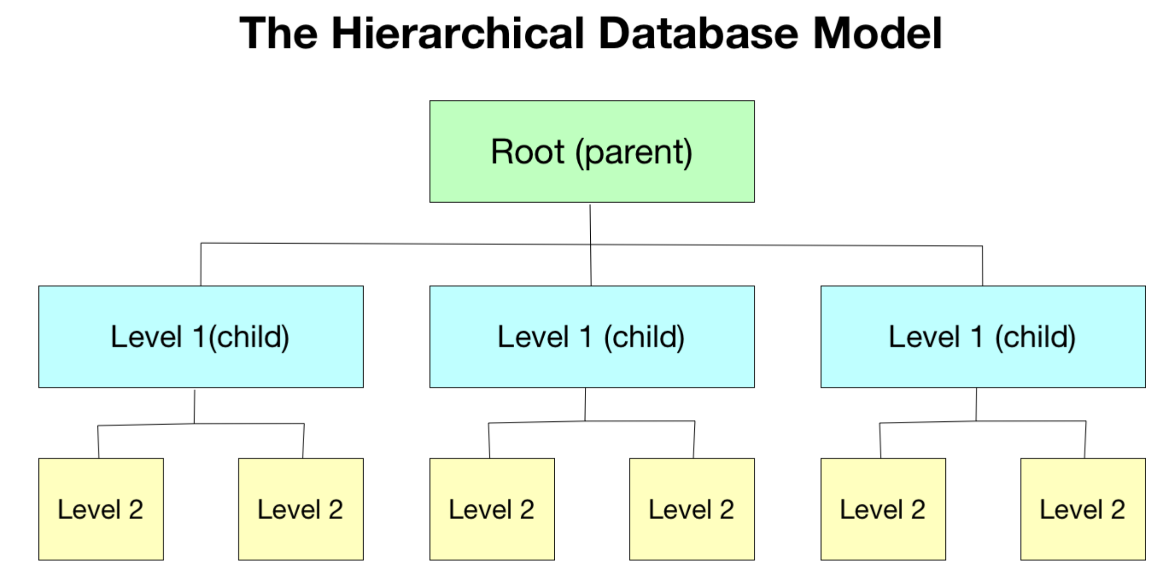

Figure 3.7: Hierarchical Model

with another node from the tree even if it isn't from the same branch. Even if this model offered improvements it wasn't massively used as IBM continued developing its products with the hierarchical model.

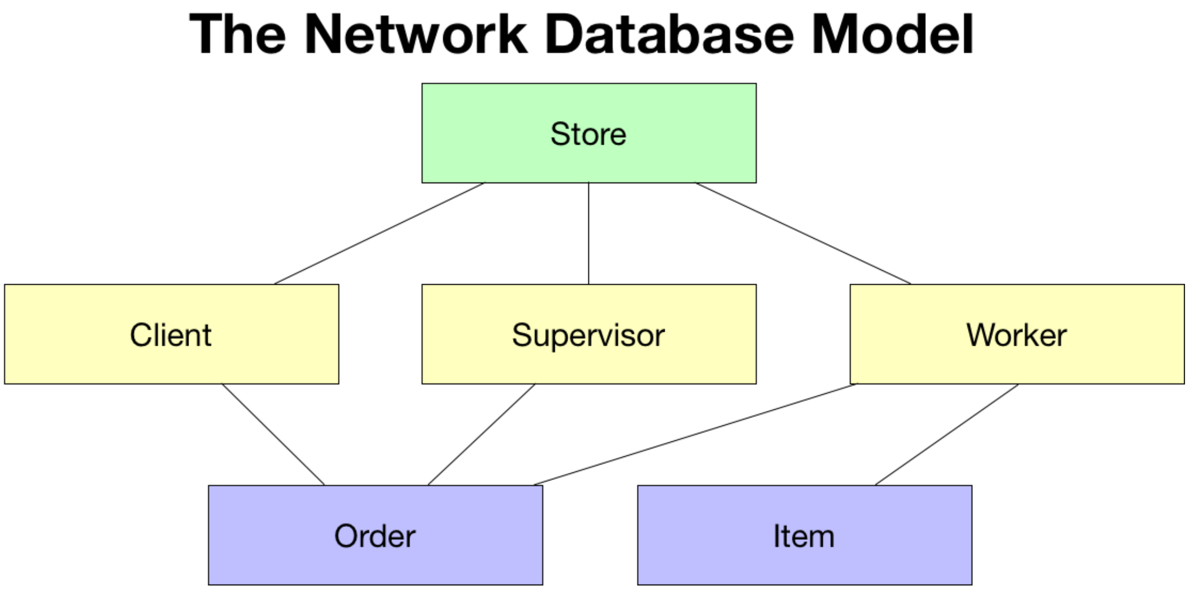

Figure 3.8: Network Model

- **Relational model:** This model was easier to understand, and the programming interface was better. The problem was that when this model came to light, at the same time as the network model, it couldn't be used due to limiting computer power, this in the 1980s was overcome by breakthroughs in the computing industry.
  This model differs from the other two models as it uses tables to store data, this

data can be related with other tables making it a very attractive solution and making it easier to maintain.
There are three ways of relating data in this model:

- One-to-One
- One-to-Many
- Many-to-Many

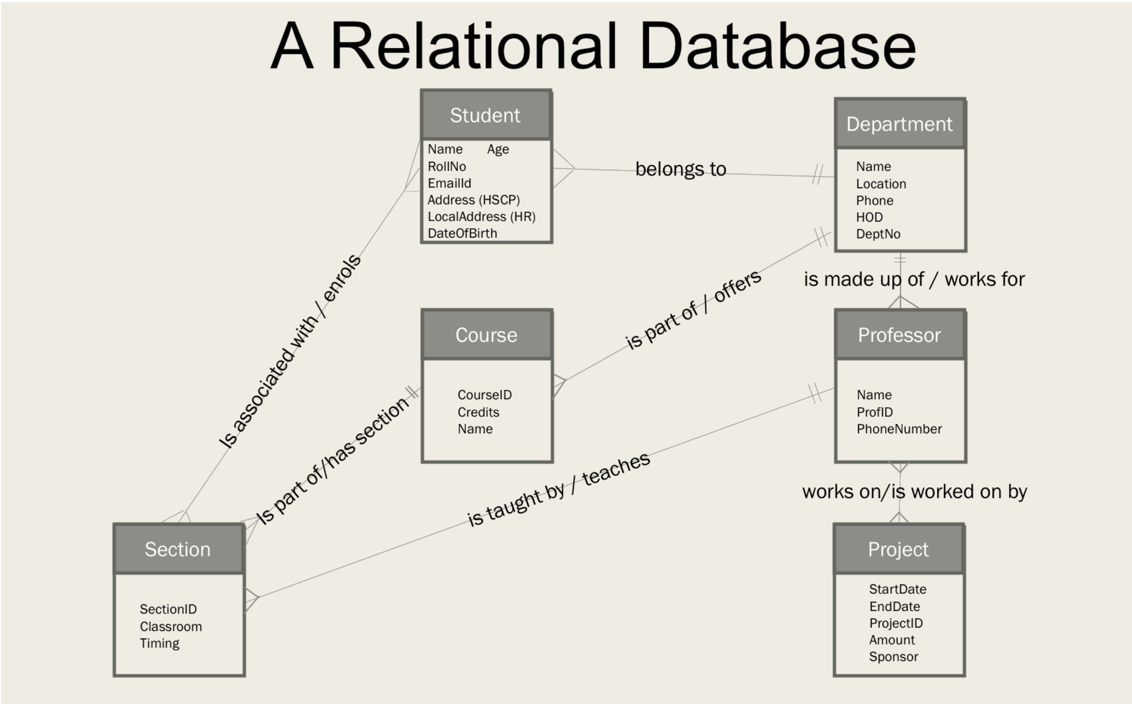

Figure 3.9: Relational Model

The language this model used is called Structured Query Language (SQL), there are several variations to this language depending on the application being used. Soon this became the standard and started being used in enterprise environments. In the mid-1990s a revolution in the development came about and new open sourced software with no cost for the developer started getting traction and with it the open source project of MySQL, which first version was developed by a Swedish company. The fact it was free, lowered the entry barrier for new developments.

- **NoSQL:** In 1998 a new term was created, NoSQL that categorized all databases that didn't follow the same language as the most common relational databases. They came about with the internet boom, this was because a shift was coming around and more and more data was being ingested by the database. SQL databases have a harder time scaling up, with more demand in internet traffic and speed becoming the focus on the server-side, new ways of storing data had to be developed. It focuses more in simplicity and maintenance as it is easier to add new fields to existing data.

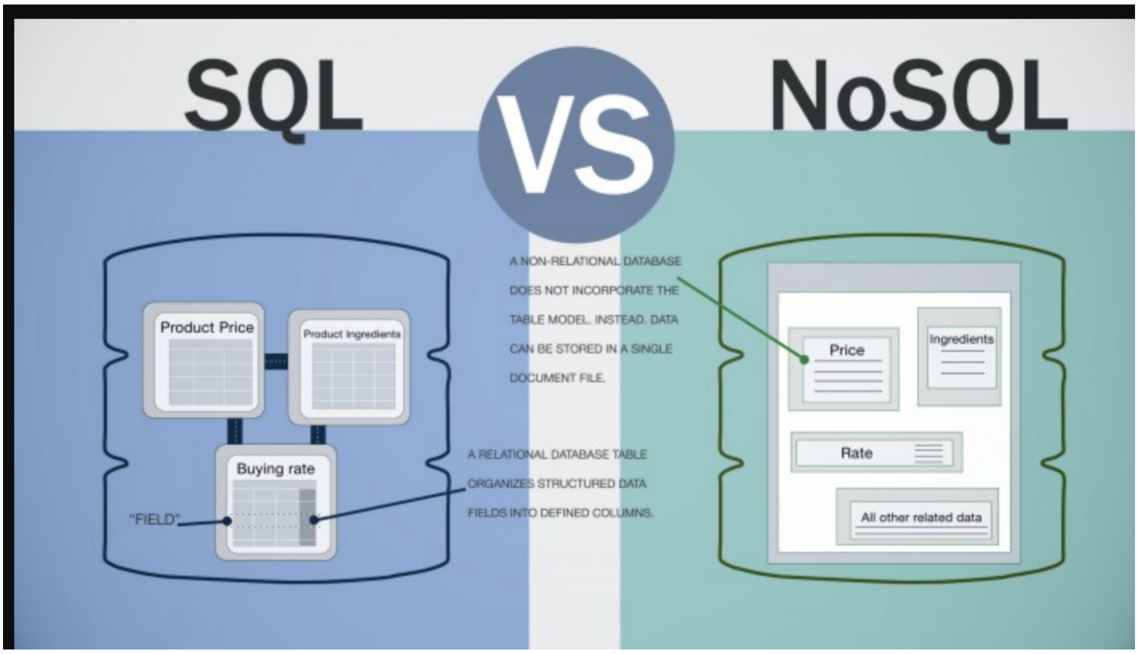

Figure 3.10: SQL vs NoSQL

Big Data has played a major role in the last few years, with storage prices in decline and the affordability of storing large amounts of data, has allowed scientists to analyze data to extract relevant information. This data is not only structured, created using applications or employees, it is data that can be extracted from unstructured sources. This can be comments from people on a product or if a media campaign is causing any impact. This information is highly valuable and that is why companies are so invested in it.[10]

3.5.2 Solutions

In this section we will expand on some of the solutions available for SQL and NoSQL databases.

NoSQL

This are some of the solutions for NoSQL databases, there is also on-premises or cloud databases that can be used.

Name	Description
On-Premise	
MongoDB	Most popular database, uses json-like documents to store data, which makes it much more expressive.
Redis	An open-source, distributed database that uses key-value to store data which makes it very fast to extract data.
Cassandra	Open-source database created by Facebook with the focus on scalability and high availability.
Cloud	
AWS	
DynamoDB	Key-value and document database that offers high scalability with database management automated.
DocumentDB	Compatible with MongoDB, offers high performance, it has the compute and storage elements separated to have more flexibility on scaling the database.
Microsoft Azure	
CosmosDB	Azure solution that allows the use of different NoSQL models with the functionality of scaling dynamically across the globe.
Google Cloud	
Bigtable	A very low latency solution that like other solutions focuses mainly on scalability and availability. Allows easy integration with big data tools.
Cloud Firestore	Offers high availability with data replication, more expensive solution than Bigtable, the main difference is that it offers better functionality for transactions.

Table 3.2: NoSQL Solutions

SQL

There are solutions both for on premises and cloud, some use the same language while others offer their own language.

Name	Description
On-Premise	
MySQL	The most popular open-source SQL database.
PostgreSQL	Focused more on the enterprise side, it offers higher extensibility than MySQL.
Oracle	Offers the full service, but comes at a high price, used more by big enterprises that can't risk security mismanagement.
Cloud	
AWS	
RDS	Amazons own relational database solution, it offers a platform available with different database engines such as MySQL, PostgreSQL, Oracle and many more.
Aurora	A database engine that focuses on a higher throughput and scalability.
Microsoft Azure	
Azure SQL Database	Their own proprietary solution for a cloud database.
Azure Database	A solution compatible with existing SQL engines such as MySQL or PostgreSQL.
Google Cloud	
Cloud SQL	A google platform to use with database engines such as PostgreSQL or MySQL.
Cloud Spanner	Solution that combines SQL elements with NoSQL.

Table 3.3: SQL Solutions

3.6 Raspberry Pi

A raspberry pi is a small single board computer with very low power consumption. Although limited in computing power its processor is still capable of running many programs and with recent models the hardware as gotten more powerful while maintaining its low price.

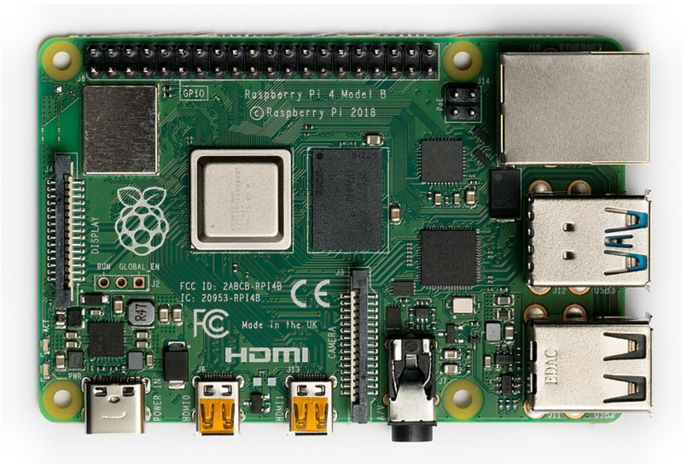

Figure 3.11: Raspberry Pi 4B

The raspberry pi has an ARM architecture processor which even though it benefits the pc on its low power requirements, it has compatibility issues with some software that is built for the x86 x64 processors.

The raspberry pi has many uses and as an affordable pc, here is a list of some examples to show the potential this device has, the limit to what you can do with a low powered computer are up to the developer's imagination.

- **Media center:** You can configure the raspberry pi to act as a place to store or load films to watch on the users tv.
- **Retro gaming machine:** It can be used to remember old retro games by using the raspberry pi as an emulator for old games.
- **Game streaming:** Another option is using it as a device for games, streaming from the pc to the tv.
- **Security camera:** As it has an IO interface the developer can add modules like cameras and us it for surveillance to keep a house safe or check on the baby.
- **Home automation:** Having access to the local network the raspberry pi can be used as the brains of the house and use it to check data on the temperature, turn the heating on and plenty more.
- **Server:** The raspberry pi can also be used as a server, although limited in computing power it still has capability to be used for projects or as a development server.

3.6.1 Clusters

A cluster is a set of computers that are tightly or loosely connected to work with each other, it acts like a single system, using the different computers to balance the load or for specific tasks.
For more intense projects these clusters may be required, the advantage the raspberry pi has is that its affordable price and small size factor the developer can build a powerful system with them. It can be used to balance the load on a web server. It can also be used as a mini supercomputer by supporting large amounts of cores and a large amount of RAM memory.

3.7 Cloud Services

Originally every company that wanted to manage their data and applications had to have a on-premises server to manage it, this required technical people having to be hired for the maintenance and upgrades the server required, when a company's business model isn't related to servers this becomes a burden. Another issue this causes is if not enough money is invested on this the availability of the server would be bad, and if it had to offer services in different countries latency becomes an issue.

Cloud computing offers a solution to these issues, a company can custom select what type of system they need to build their infrastructure, this can adapt to the companies needs in relation to processing power or storage space, this means that if a company focuses in the storage and transfer business they can build an infrastructure that focuses more on capacity, transfer speed and storage speed, compared to another company that works with artificial intelligence that may need a more powerful computer and less storage space. This becomes available thanks to the improvements on virtualization.

3.7.1 Virtualization

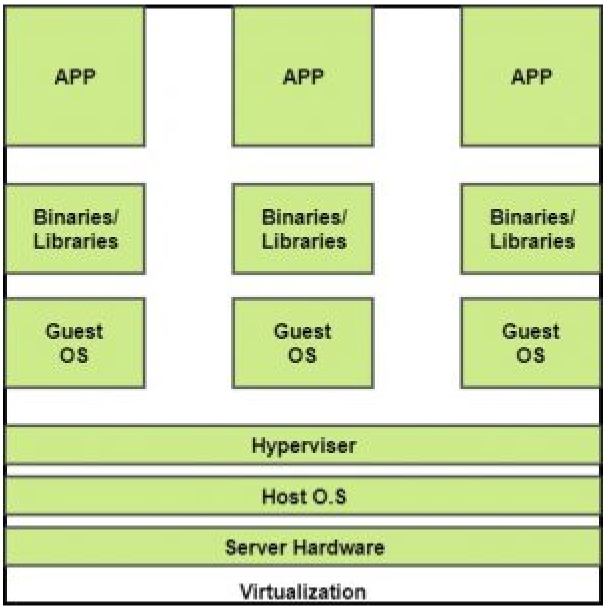

Figure 3.12: Virtualization System

Virtualization is a technique that separates the physical layer from the service level, this allows that a computer with certain resources have several virtual operating systems on top with different resources allocated. This offers many benefits in relation to cloud computing.

- Dynamic allocation of resources
- Lower costs of infrastructure
- Rapid scalability and remote access
- Multiple parallel OS
- Pay for time resources used

There are different types of virtualization for different parts of the computer:

- **Storage virtualization:** This allows several servers storage as a single system, this virtualization allows it to be redundant as it offers tools for data replication and recovery, so if a server fails or a hard drive gets damaged the data remains intact.

- **Desktop virtualization:** This virtualization allows the user of the virtual system to access it remotely without knowing in what machine the OS is located.
- **Network virtualization:** The ability to create virtual networks to connect different elements in the data center allows different services to be able to communicate with each other.
- **Application virtualization:** With this type of virtualization different versions of a same application can coexist and allows an application to run without being installed in the traditional way or be supported by an OS.

3.7.2 Providers

- **AWS:** Amazon Web Services launched on 2006 and pioneered the business model for on-demand cloud computing, it has the highest market share of all the providers available. It offers a very large selection of products to choose from, making it better to create a well-defined architecture. AWS focuses more in the public cloud making enterprises with their own data center hard to interoperate with AWS.
- **Google Cloud:** The underdog in the competition between AWS and Microsoft Azure, it doesn't have an enterprise focus which makes it harder to attract enterprise clients. It provides services with deep learning and machine learning which google cloud has leading technology.
- **Microsoft Azure:** A close competitor from Amazon Web Services, with years of experience and close relations with many enterprises Microsoft Azure offers what these clients want, that is to operate Azure functionality mixed with their own data center, allowing for a hybrid cloud.

3.8 Testing

Testing is an essential part of software development which allows code to be bug free, even though it may seem tedious at the beginning in the long run it pays off and other developers will appreciate a built testing infrastructure.
There are several types of testing:[11]

- **Unit tests:** These are the lowest level of testing, where individual methods and functions are tested to verify that they are working correctly. Unit tests are easy to automate and are quite common in continuous integration, where in every new build it verifies that anything previous is working correctly.
- **Integration tests:** These tests are higher level where different modules or services work well with each other. The complication of running these tests is that it requires certain parts of the application to be running.
- **Functional tests:** Focusing more on the business requirements, these tests only are interested in the result not the middle steps between starting the action and the result. It differentiates from integration testing is that integration tests are only interested in checking if the system works and functional testing is more interested in verifying that the result is correct.

- **End-to-end tests:** End-to-end testing simulates user behavior to test if the flows of the application are working correctly. It is a very high level of testing and it is more complicated to automate due to needing interaction directly with the application, this interaction if automated needs to be maintained because any update on the interface of the application can render the test useless.
- **Performance testing:** This type of tests is very important to have when building big applications as it provides relevant information on how scalable the application is and if any significant latency or crashes are noticed.
- **Smoke testing:** Like functional tests but less expensive, just to verify the main systems are operational, if the deployment is functional then more expensive tests can be run.

3.8.1 Automation

Having a person run the tests above is possible but it would be very expensive, counter-productive and a very boring job as they must run every time a deployment or a new feature is added. To automate the tests programmatically the developer needs a testing framework that suits the application. These frameworks are dependent of the programming language used. Even though this automation allows to execute tests to verify the application the next step is to whenever the developer chooses to deploy the application, the tests run automatically before deploying, this is called continuous integration.

CHAPTER 4

Solution Proposed

4.1 Description

The system developed uses several elements in conjunction to work correctly, the core of the system is the actual AI, which has the name of Sam as it offers a unisex way of talking with the chatbot. In this chapter we will go into detail on how the system has been built and how the different elements provide the support for the different features the bot has.
The objective of this system is to create a simple way of tracking your progress, create dynamic tables which learn from you and provide a correct assessment and guidance to reach the users objective. The following points reflect what a final solution features would be:

- **Diet table creation:** Like the exercise table, this diet table would be based on what objective the user wants to reach. The bot will also consider what allergies the user may have and try to progressively adapt the table in order to transition from the user's normal diet to the new diet without altering it all at once.
- **Training sessions:** This is a feature that could be added in the future if the speech technology keeps improving, as it would be ideal to have headphones on while Sam guides the user through the training session, this would make the user use less the phone and still keep track of brake times.
- **Gym location:** If a gym franchise or a multiple gym service bought the application this feature could make it easier for the user to find a gym nearby or maybe check what gyms are available at a certain location.
- **Exercise tutorials:** This could be a very useful functionality to make it easier for the user to know how to do an exercise, as some exercises by name may be confusing and this will make it easier to find out, compared to looking for it online.
- **Progress tracking:** With this the bot and the user can have an idea of how the training sessions is progressing, based on the results from the users measurements

the bot may choose to change the users diet, exercise table or remind the user to follow the diet table given.

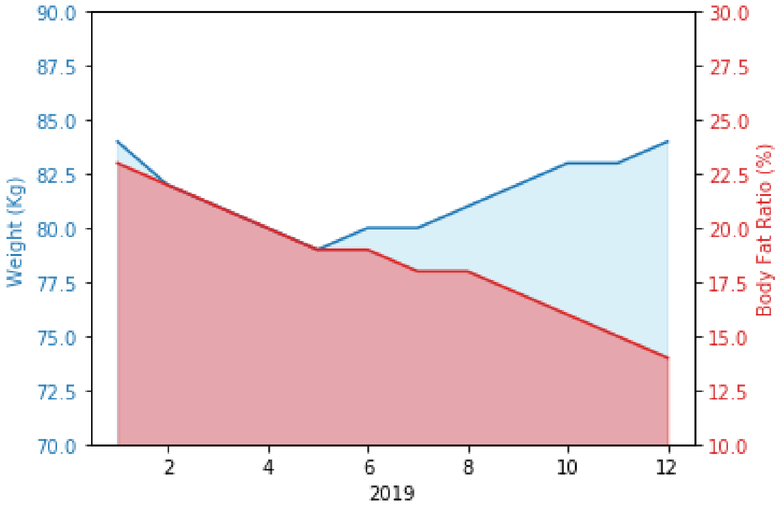

Figure 4.1: User Weight vs Body Fat Ratio

- **Exercise table creation:** The bot provides new tables every month or may change if it sees throughout the month very little progress from the user. This table would be designed to fulfill the objective the user wants.

WORKOUT #1	Sets	Reps	Tempo	Rest
1. Assisted Chin-Ups	4	8	2011	60-90s
2. Flat Dumbbell Press	4	8	2011	60-90s
3. Seated Cable Rows	4	8	2011	60-90s
4. Seated Dumbbell Shoulder Press	4	8	2011	60-90s
5. a) Abs exercise of choice b) Abs exercise of choice	4 4	8 8	1010 1010	-- 60s
WORKOUT #2	**Sets**	**Reps**	**Tempo**	**Rest**
1. Barbell Squats	4	8	2011	60-90s
2. Alternating Dumbbell Lunges	4	16	2010	60s
3. Stiff-Legged Deadlifts	4	8	2011	60-90s
4. Standing Calf Raises	4	12	2010	60s
WORKOUT #3	**Sets**	**Reps**	**Tempo**	**Rest**
1. T-Bar Rows	3	15	1010	45s
2. Incline Dumbbell Press	2	15	1010	45s
3. Close-Grip Lat Pulldowns	3	15	1010	45s
4. Push-Ups	2	15	1010	45s
5. Machine Rows	3	15	1010	45s
6. a) Bent-Over Dumbbell Raises b) Lateral Raises	2 2	8 8	1010 1010	-- 60s
7. a) Abs exercise of choice b) Abs exercise of choice	3 3	12 12	1011 1011	-- 60s
8. a) Side Plank b) Plank Hold	3 3	20 sec 20 sec	-- --	-- 60s
WORKOUT #4	**Sets**	**Reps**	**Tempo**	**Rest**
1. One-Legged Squats	2	15/side	1010	60s
2. Dumbbell Split Squats	2	15/side	1010	60s
3. Full (Rock-Bottom) Dumbbell Squats	3	15	2010	60s
4. Walking Dumbbell Lunges	3	20 steps	1010	60s
5. Bodyweight Calf Raises	3	20-30	1010	30s

Figure 4.2: Exercise Routine

- **Route recommendations:** At the moment the core functionality is based on gym training, especially weight lifting, but an interesting functionality that could be added in the future is for outdoor bike and running routes. With recommendations from users on the best routes.
- **Team meetings:** This functionality focuses on the social part of training, where users can arrange team matches between the users in what interest them more, even if the user's interests are outdoor training or cycling teams, where they can arrange a route up a mountain, this will provide a community feeling in the application.

4.1.1 Reason for Building the Application

To prove that the application has a market opportunity an inquire was made to a group of people with over 200 people, the results where interesting as will be explained below. The language of the form is in Spanish as the public inquired live in Spain.

As shown below most of the users filling the form are from age 16 to 25, this gives context to some of the answers given afterwards, in this case generational mentality plays an important role as different age groups may have a different opinion on gym training. In this case the target group even though available for every age group is more focused on people in those age limits as they are the most common at gyms, the least informed and the more friendly with technology they are.

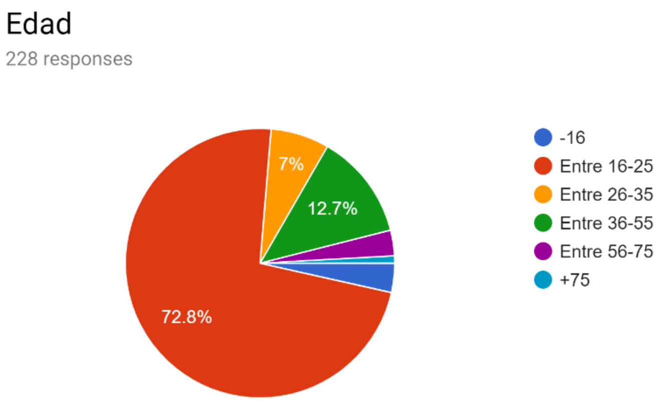

Figure 4.3: Survey: Age

In the case of gender, the pool has been varied with close to 50% of each gender, this with the age group gives a general perception of what people think of the questions asked.

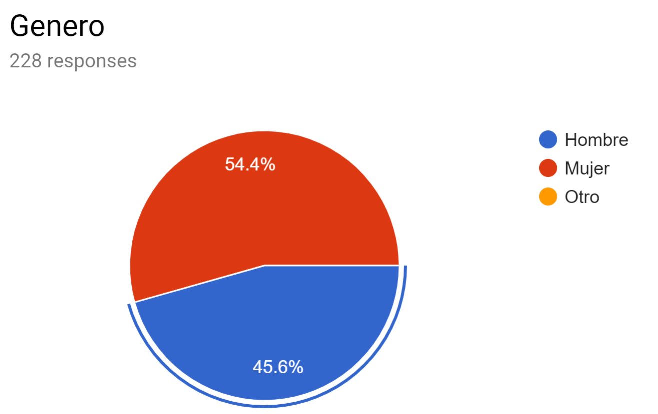

Figure 4.4: Survey: Gender

The following graph starts to show some interesting results, we can see that 30% of the people asked don't do any type of training and a sum of 24,1% only train one to two times a week, this isn't unexpected but that is over 50% of people that train below the recommended amount weekly.

Figure 4.5: Survey: Gym Frequency

In the next graph the conclusions we can extract is that from the people that did the survey 30,7% follow their own knowledge when going to the gym, this is something that is very frequent in gyms and it is the main source of dropouts as many people that follow what they know don't actually have the required knowledge to do so. Another point we can extract from here is that 11,4% use tables while training, this could provide a potential group to focus the bot to. The inconvenient part of information we can extract is that people don't usually use applications when they train, so in order to reach the audience, the bot must seem as human as possible.

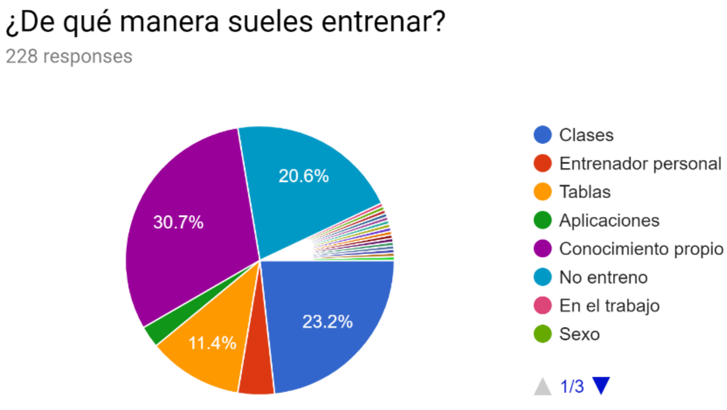

Figure 4.6: Survey: Training Type

The following pie represents if a personal trainer would motivate a user to go to the gym more often, this was surprising, as over 50% would be motivated to go to the gym if they had a personal trainer guiding them and 34,2% would maybe feel motivated to do so, after analyzing the previous graphs we consider that there is a market for a personal trainer application.

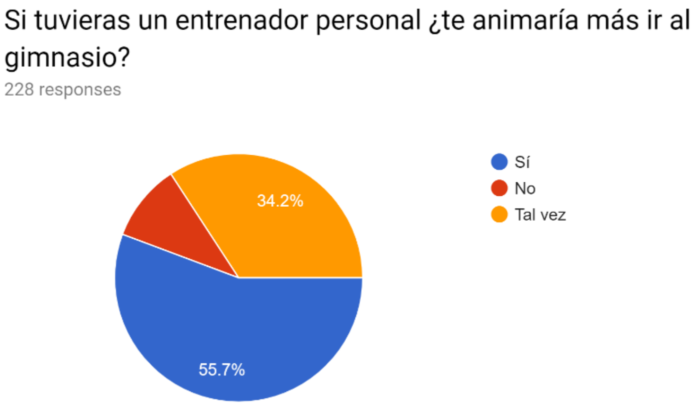

Figure 4.7: Survey: Having a Trainer

In the chart below we can extract some conclusions from the data, we think that 64% of users change in some way the way they eat when they start training, but most don't follow a strict diet. This is an important factor when developing the bot as the functionality for the recommended diets can be adapted to provide diets which tries to follow the most common eating habits of the user and slowly transition to a stricter diet, this will make users less prone to not following it.

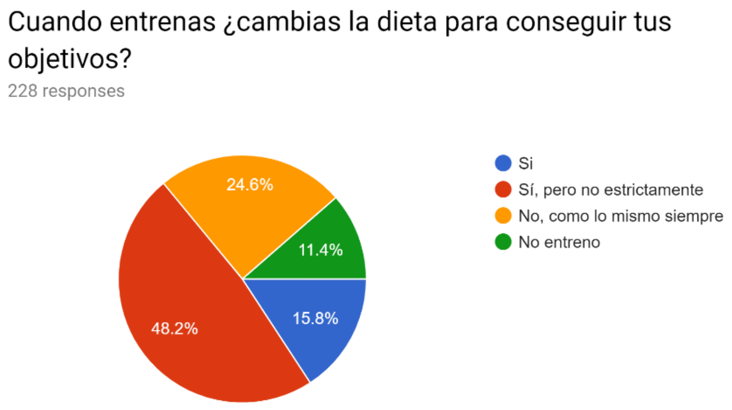

Figure 4.8: Survey: Diet Change

This graph follows up with the previous one and supports the conclusion extracted, that users rather have someone assisting with what meals to recommend the user with 84,6% that would follow a diet recommended to them.

Figure 4.9: Survey: Diet Recommendation

The same as the previous chart in relation to exercises, the public is open and prefer someone guiding them to reach their objectives, in this case regarding exercises. This has to do with people using their own knowledge to train as compared to this graph some of the people that followed their own knowledge would rather have some one recommend what exercises to do.

The most important chart of all is regarding our application, which is if these recommendations where given by a chatbot, if they would still follow it. Some curious points can be extracted from this pie, which are that most people would rather hire a

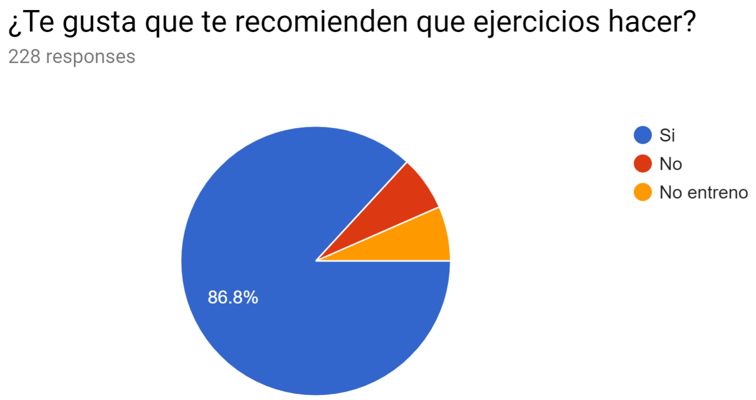

Figure 4.10: Survey: Exercise Recommendation

professional trainer, what this tells us is that many people still prefer human contact over having a bot telling them what recommendations to follow, another point which links to a previous graph where we talked about how people train is that only 4.1% have personal trainers, the reason being that hiring a personal trainer is costly so considering that and the fact that 35,1% of people would follow those recommendations gives the application a large market to focus to, but also implies that the chatbot must try and simulate a human being as good as possible.

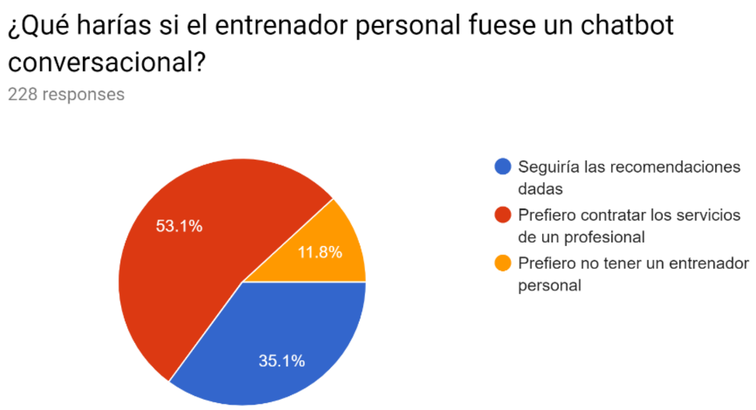

Figure 4.11: Survey: Bot Trainer

4.2 Design

In this project we find that there has been two different designs, the original design was developed in a way to make the system homemade, without depending on an external platform which even though it provides a base where to start developing it limits the control the developer has in adding new functionality.

The second and final design does use an open source platform as the base to start developing, this had to be done because of the complexity related in developing the proprietary solution in relation to the time available to develop it. Further down the memory the reason for changing the design will be explained in more detail.

4.2.1 Original Design: Frontend

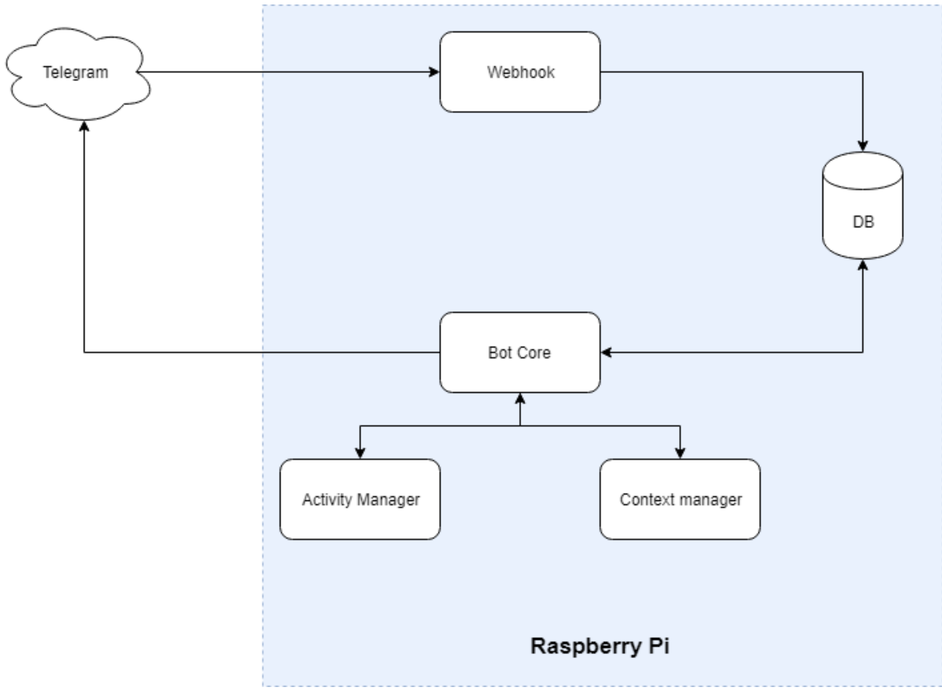

Figure 4.12: High Level Original Design

Following a model view controller architecture, the project uses as the view part Telegram, a chat platform for the users to contact the bot. The bot can be ported to different chat platforms based on preferences from the users, the reason Telegram was chosen instead of another platform where:

- **Higher user count:** Compared to Slack, another chat platform with support for bots, Telegram had for the year 2018, 200 million active users, while Slack only has 10 million. The reason to choose a chat platform with a higher count for active users is because with more users it is easier to penetrate the market. As the graph shows below it is also important that Telegram has been steadily growing in recent years with an increment of 100% in one year from 100 million to 200 million active users.

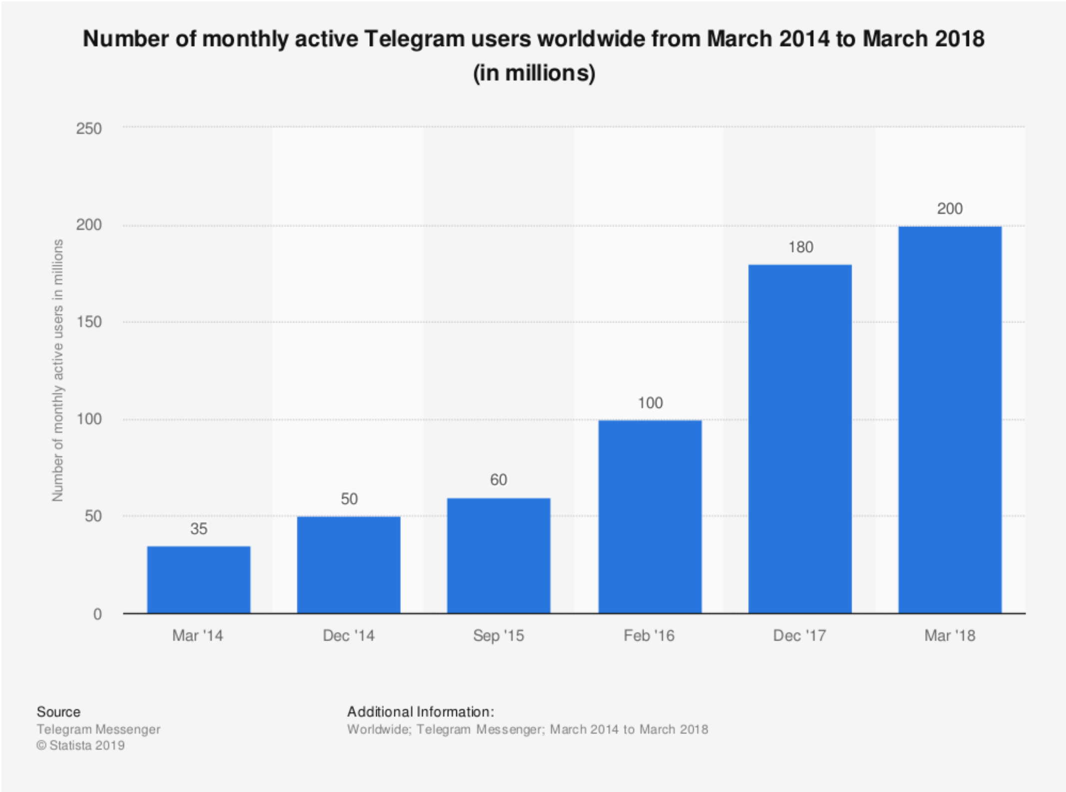

Figure 4.13: Telegram User Growth

- **API:** Telegram has a very well documented API which makes it easier to integrate with the bot, also with previous knowledge in developing with Telegram it was the logical choice to use.
- **Webhook:** It provides a way of creating a bridge between Telegram servers and the Raspberry Pi to reduce stress at the network as Telegram automatically sends messages to the server instead of relying on the Raspberry Pi, polling Telegram servers.

Webhook

After analyzing options to get the messages from Telegram, it was decided for this design to have a webhook instead of polling the server. A webhook is the bridge for communicating with the Raspberry Pi allowing to relieve the network from redundant traffic. The way a webhook works is like a web server which Telegram calls it whenever a message is received from a user. Whenever the webhook receives the message, it stores it in the database where the bot can check if anything new has arrived, this allows to transfer the stress from the network to the pc which reduces latency when receiving messages as polling a server takes longer than polling a database.

4.2.2 Original Design: Backend

As the frontend was mostly managed by Telegram there wasn't much to design except the webhook to support the message delivery.

For the backend we find the controller and the model for the bot. The way this two were designed is in that the controller was the communicator, were it managed the database where it extracted the user's data two give back to the functions that performed the tasks. The first thing the controller does when getting a message from the database is authenticate the user, this is done with the telegram id sent from telegram, this id is unique to each user which makes it secure to identity fraud, at least from the applications end.

Below is the original flow for a message sent by the user to the bot, the design is at a high level to see where the message goes through until it gets to the response.

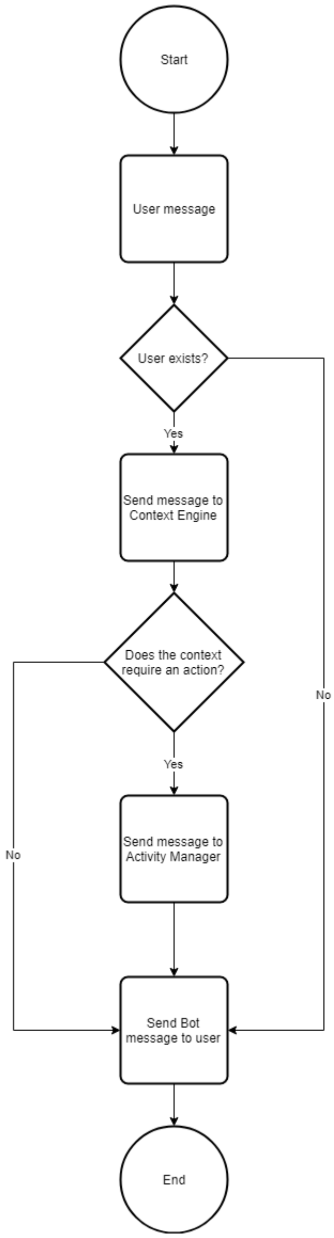

Figure 4.14: Main Flow Original Design

The flow follows starts once the message is grabbed from the database, the first thing is to verify that Sam is talking to a user from the platform, the idea is that the user doesn't belong to the platform Sam offers to join the club. The complex part of the application was the context engine side of it.

Context Manager

The context engine is an engine which objective is to give the bot context from the user, this gives intelligence to the bot as it can remember what has been talked before. It also has the function of extracting data from what the user's message. The following flow shows how it is intended to work. These are the main design objectives when creating the context engine.

- Being able to complete actions without following the happy path structure, in other words, if the bot wants to add measurements and there are two different measurement types there isn't one way to provide those measurements, the user can facilitate them in several ways.
  - **I want to measure myself:** In this case the bot then will ask for the user's weight.
  - **Sam please add 80 kg as my new measurement and my current body fat ratio is 19%:** In this case the context engine will extract both variables and delete the context as all relevant data has been extracted.
  - **Sam my new body fat ratio is 19%:** to which Sam will detect that the user has said its body fat ratio but not his weight and will ask the user to provide his weight as well.
- Keep the user in track whenever the user deviates from what is being asked.
- The context engine must be capable of extracting data correctly.
- Scalability is an important objective that the context engine must be designed for.
- The flows can be generated without having to manage code directly, the idea to make this work is to have the flows standardized as JSON formatted file.

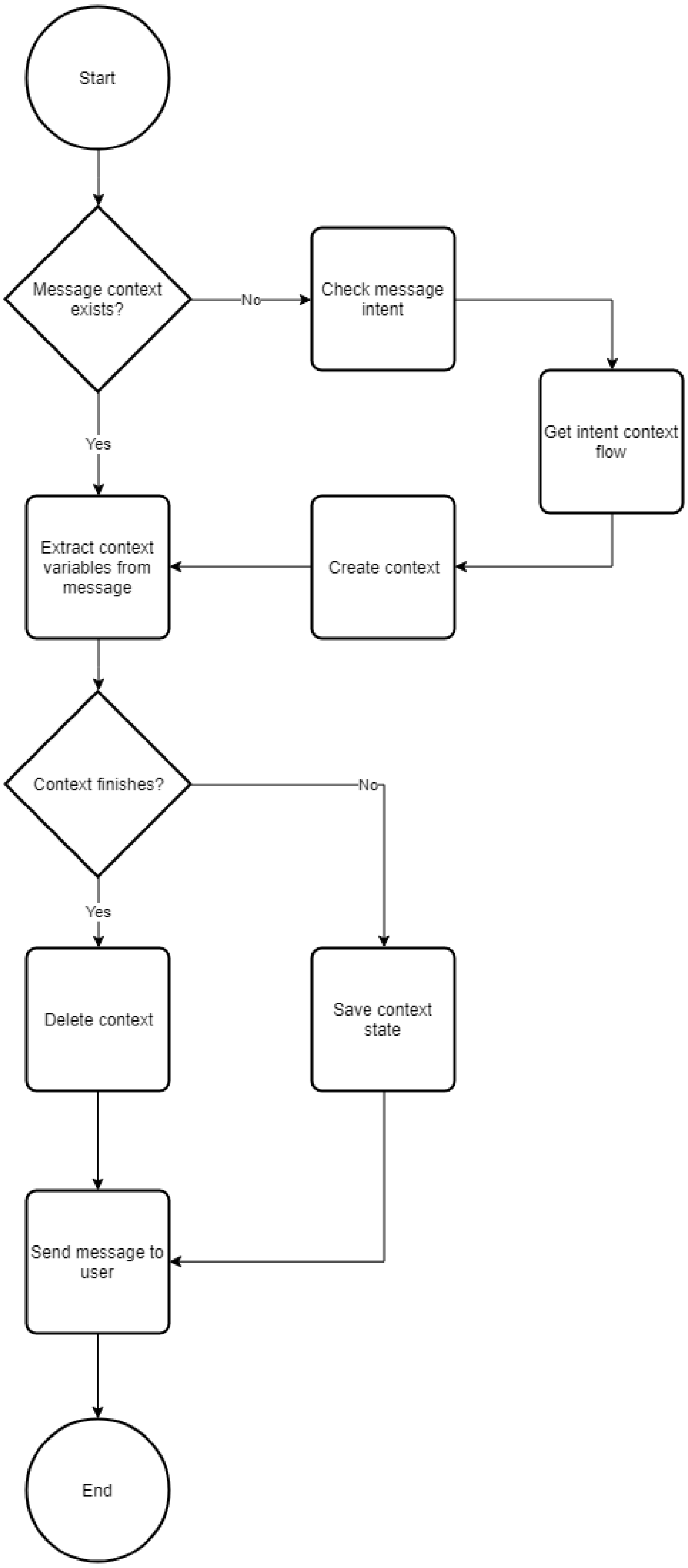

Figure 4.15: Machine Learning Original Design

Activity Manager

The activity engine is the one in charge of performing the tasks required for the bot to be functional, the design is simple but the complexity in this manager is the functionality the developer wants Sam to have. The design for this is that it must be simple to add new functionality and not have to change to much code. The idea is to use the flows we talked about in the context manager section, in these flows if the context requires an activity to have the name of such activity in the JSON file and execute it in the activity manager.

Scalability

An important requirement for Sam is for him to be scalable, to reach this objective the architecture is designed in modules so that if a certain part of the application is receiving a higher level of stress to be able to scale it independently. This is done by having two masters, one for the activities and one for the context manager, these when launched by themselves act as a normal manager, but when several are launched acts as a load balancer.

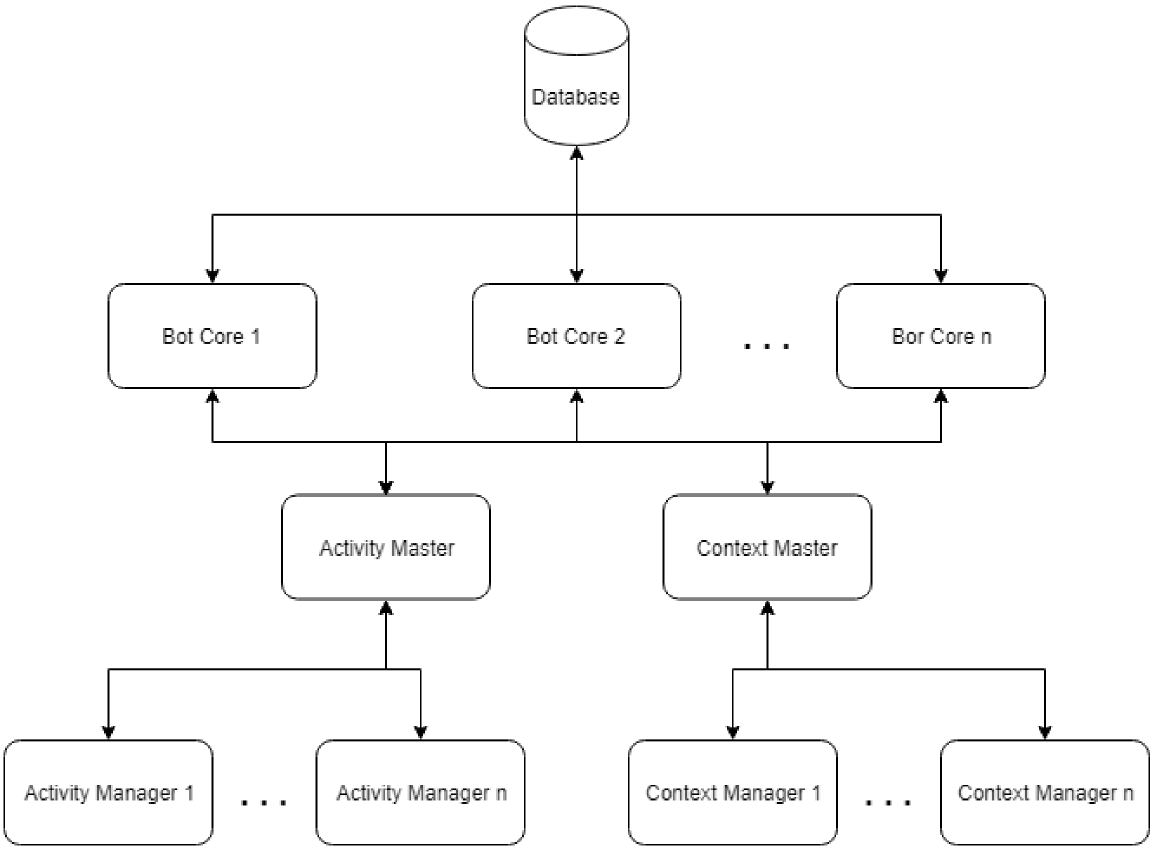

Figure 4.16: Scalable Structure for Original Design

4.2.3 Final Design: Frontend

For the final design the frontend only received some changes, the new design, as the original, still uses Telegram as the chat platform. One aspect that changed was the webhook, which for the revised version comes integrated with Rasa, so it doesn't make any sense to develop it.

4.2.4 Final Design: Backend

As everything is designed following some rules, the changes the backend can have is limited to what Rasa offers.

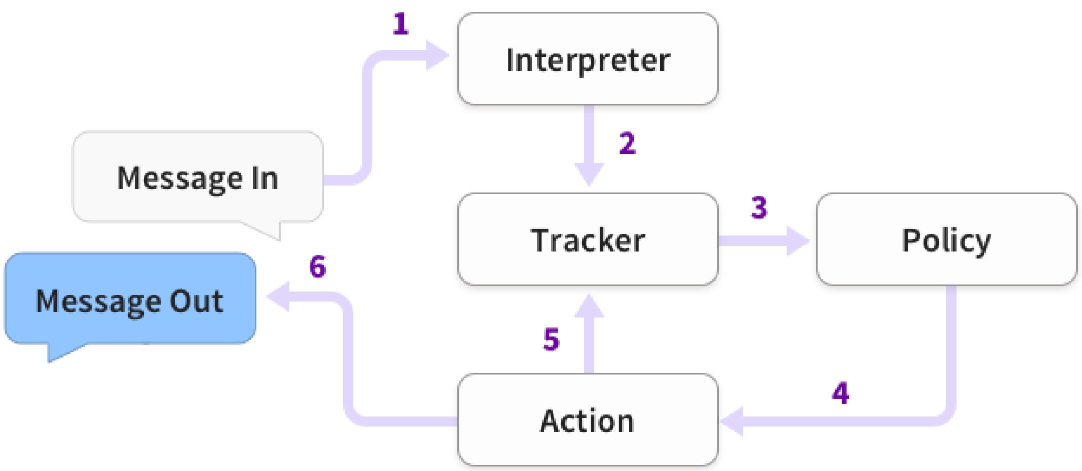

Figure 4.17: Rasa High Level Structure

The steps this new backend follows is different.

1. When a message is received it goes to the Interpreter, which in the original design would be part of the context manager, in the interpreter a dictionary is created which contains the original text, entities and the intent.

2. An object linked to the state of the conversation called Tracker receives the info that a new message has arrived.

3. The policy receives the state of the conversation with the user, this information comes from the tracker.

4. Depending on the state the policy choses what action to take.

5. The tracker logs the action taken from the bot.

6. The response is sent back to the user.[12]

Activity Manager

The activity manager has been changed because of the implementation of rasa, the way it has been designed is with what is called custom actions, which is similar in the way it was intended in the original design, but differs in the way it interacts with the data as it uses the database in conjunction with the Tracker from rasa which stores all the data extracted from the conversation.

Story Manager

The way that Rasa manages conversational chatbots is with the creation of stories. The stories are basically the flow of how the conversation is going to work as explained in section 3.4.2 in the state-of-the-art chapter. The advantage of this design compared to the original one is that it manages better deviations from the happy path, which is the correct way of completing a story. The disadvantage is that it requires a lot of training conversations for the bot to perform as intended, the design in the original had more restricted paths, so it was harder for the user to deviate from the original flow.

4.2.5 Database

The database remained the same for both designs as both required access to the same data. The design for the chatbot initially remains simple as the functionality to be developed for this proof of concept is limited. But the focus is around the individual user and with further development the addition of tables for groups and other more complex functionality may be added. Th use of a relational database was chosen as the preferred choice as the data's integrity is relevant and queries are a must for this application.

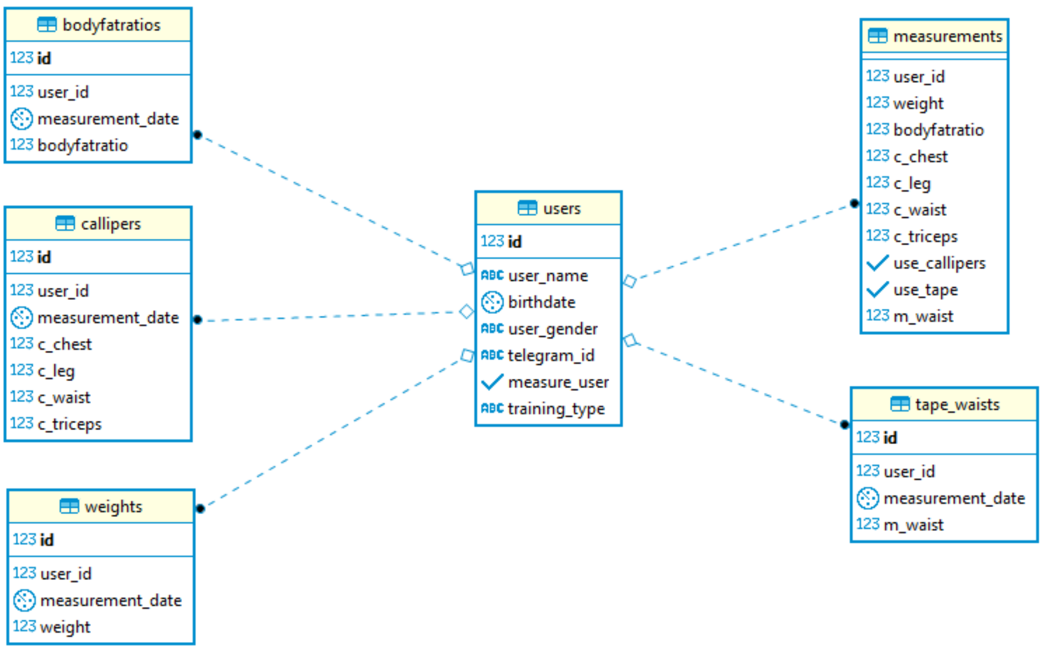

Figure 4.18: Database structure

The language to manage the database has been PostgreSQL as the team has had more experience developing in that language and the possibility of having more extensions than MySQL may offer a better option to have in the future.

4.3 Resources Used

4.3.1 Hardware Resources

- Raspberry Pi to act as the server of the application, the brains of the chatbot, for more information check section 3.6
- Azure as the cloud provider after porting the solution to the cloud, for more information check section 3.7.2
- Smartphone to test the bot and to use the application
- Laptop to develop the code and manage the server.

4.3.2 Software Resources

- Python programming language
- Rasa Core for the second design, for more information check section 3.4.2
- Atom, as the editor where the code is developed
- GitHub to keep track of the code
- NGINX to tunnel the web traffic received in the server to the correct port
- PostgreSQL as the SQL language to query the database, for more information check section 3.5

4.4 Implementation

4.4.1 Original Implementation

Webhook

To develop the webhook first we had to prepare the Raspberry Pi to be able to act as a web server, as it was connected at home some management to the router had to be made in order to allow inbound traffic for HTTPS as this is the standard Telegram requires for the webhook to work. For it to be secure it was also required to create the certificates needed for SSL. These certificates where created with OpenSSL. Another issue that had to be solved in order for the webhook to work is have a domain name, the reason this is needed is because the network provider gives IP's dynamically which caused Telegram to lose the connection whenever the IP's was reset, initially a method was developed to automatically detect whenever the IP had changed, this was a very hacky way of solving the problem which resulted in some issues where the algorithm failed to update the SSL certificates which resulted in data loss and required human intervention to fix. To solve this problem a dynamic domain name provider by the name of `www.noip.com` provided a free DNS service which automatically updated the DNS whenever the IP had changed.

Once the Raspberry Pi was set, it needed to have the web server running in order for it to work, below is the code for the API Rest that Telegram has to use in order to send the message to the server, has you can see once the message from Telegram is received it is stored in the database, this code can be expanded to support multiple platforms.

```
from flask import Flask,request,abort
import json
import psycopg2

try:
        conn = psycopg2.connect("dbname='pt' user='matthew' host='localhost'
            password=***")
except:
        print ("ERROR: Unable to connect to the database")

app = Flask (__name__)

@app.route('/wbhk/tlgrm', methods=['POST'])
def webhook():
        if not request.json:
                abort(400)
        cur = conn.cursor()

        cur.execute("INSERT INTO webhook.incoming_messages (message, source)
            VALUES ('"+json.dumps(request.json)+"', 'Telegram')")

        conn.commit()
        cur.close()
        return json.dumps({'success':True}), 200,
            {'ContentType':'application/json'}

if __name__ == "__main__":
        app.run()
```

Controller

What the bot does at this stage is ask the database if any new message has been received, in order to make it scalable each bot has a unique id, this way whenever a bot gets a message from the database it can inform the rest of the bots what message is being worked on by which bot.
The bot after receiving the message sends it to the classifier through an API. Due to

changing the implementation, the user detection wasn't created, which would be used to identify if the user is in the system. After receiving the response from the context manager, it sends the response back to the user.
During the implementation of the original design the activity manager wasn't built due to changing the whole model.

Context Manager

When implementing the context manager, it was decided to create one predetermined flow for new users, this is done as we don't want the user to have access to the rest of the flows. When developing the context manager, for the sprint being done the objective was to have the intent detected based on what the user says. This was done with the code below which follows these steps:

1. Stem the user's phrase with NLTK's Snowball stemmer
2. After it converts the stemmed words into numbers.
3. Then converts the count for the words into an array of the frequency in which the words appear in the training data.
4. The different classifiers then predict what intent it is and take a vote on what intent it is, after that the result and probability that the prediction is correct is sent back to the user.

```
def classify(data):
        if "text" not in data:
                abort(400)
        global threadLock
        array_statement = [data["text"]]

        array_statement = [" ".join([snow.stem(word) for word in
            sentence.split(" ")]) for sentence in array_statement]

        # thread lock
        threadLock.acquire()
        frases_new_counts = count_vect.transform(array_statement)
        frases_new_tfidf = tfidf_transformer.transform(frases_new_counts)

        vote = []
        pred_multi = multinomial_small_clf.predict(frases_new_tfidf)
        prob_multi =
            (multinomial_small_clf.predict_proba(frases_new_tfidf)[0][pred_multi[0]])
        vote.append(pred_multi[0])

        pred_sgdc = sgdc_small_clf.predict(frases_new_tfidf)
        prob_sgdc =
            (sgdc_small_clf.predict_proba(frases_new_tfidf)[0][pred_sgdc[0]])
```

```
vote.append(pred_sgdc[0])

pred_log = log_reg_small_clf.predict(frases_new_tfidf)
prob_log =
    (log_reg_small_clf.predict_proba(frases_new_tfidf)[0][pred_log[0]])
vote.append(pred_log[0])

pred_lin = lin_svc_small_clf.predict(frases_new_tfidf)
prob_lin =
    (lin_svc_small_clf.predict_proba(frases_new_tfidf)[0][pred_lin[0]])
vote.append(pred_lin[0])

pred_forest = forest_small_clf.predict(frases_new_tfidf)
prob_forest =
    (forest_small_clf.predict_proba(frases_new_tfidf)[0][pred_forest[0]])
vote.append(pred_forest[0])

threadLock.release()
prob_sum = prob_forest + prob_lin + prob_log + prob_sgdc + prob_multi
prob_clf = (prob_sum)/5 * 100
# thread unlock
try:
        decission = mode(vote)
except Exception as e:

        log("Two intents possible", LogTypes.LOG_WARNING)
        c = Counter([1, 1, 2, 2, 3])
        c.most_common(1)
        decission = c[0]

confidence = (vote.count(decission)/len(vote)) * 100
probability = (confidence + prob_clf)/2
intent = intents[decission]
print("Probability: ", probability, file=sys.stderr)
print("Category: ", intent, file=sys.stderr)

if probability > 70:
        json_msg = {'intent': intent,
                            'sentence': data["text"]}
        new_entry(json_msg)

result = {'intent_id': format(decission),
                'intent': intent,
                'probability': "{:.2f}".format(probability)}

return json.dumps(result)
```

After reaching this objective, we started realizing that to finish creating the context manager would be to complex and being in the time constraint and haven't developed any functionality for the personal trainer it was decided to change the platform to something higher level but with still the capability of modifying it to the objective of the bot. It was a hard decision but in order to finish in time the correct choice.

4.4.2 Final Implementation

For the final design most of the code done was scraped as it was already included in Rasa.

Raspberry Pi to Azure

When trying to use Rasa as the new engine in the Raspberry Pi it started failing as the architecture in Rasa is more resource heavy, so in order to develop the new design with the credits offered by Microsoft for being a student we ported everything to the cloud. For this we created a virtual machine where Sam will be running.

Config

The config file for Rasa is where the developer designs how the bot is going to be, in this case the pipeline follows a modified version of the pretrained_embeddings_spacy, check section 3.4.2. The pipeline translates to the following steps.

1. First, the language to be processes is chosen, in our case it is English.
2. After it goes through Spacy's Natural Language Processor, where the user's message is normalized.
3. Then the message is tokenized with Spacy's Tokenizer.
4. After Spacy's features for intent classification are added.
5. The regex featurizer gets regular expressions defined in the training data.
6. The CRFEntityExtractor uses conditional random fields to extract entities.
7. EntitySynonym converts synonyms to a defined value, very useful to convert different ways to say something in a normalized form.
8. After the intent is detected using the Sklearn Intent Classifier.
9. After detecting the intent Spacy predicts with its entity extractor what entities need to be extracted, it uses the statistical BILOU transition model, it is pretrained so doesn't require new data, it is very useful for extracting user's names it is an addition to the predefined pipeline.
10. Finally, the DuclingHTTPExtractor that is also an addition to the predefined pipeline, the reason to add this to the implementation is that it provides good entity extraction for time entities such as birthdays.

```
language: en
pipeline:
- name: "SpacyNLP"
- name: "SpacyTokenizer"
- name: "SpacyFeaturizer"
- name: "RegexFeaturizer"
- name: "CRFEntityExtractor"
- name: "EntitySynonymMapper"
- name: "SklearnIntentClassifier"
- name: "SpacyEntityExtractor"
- name: "DucklingHTTPExtractor"
  url: "http://localhost:8000"
  dimensions: ["time","distance", "number"]
  timezone: "Europe/Madrid"

policies:
- name: MemoizationPolicy
- name: KerasPolicy
- name: MappingPolicy
- name: "FormPolicy"
```

Another important part of the config file are the policies Rasa follows in the flow of the conversation. The reason these for where chosen where for the following reasons.

- **Memoization Policy:** This policy just memorizes the conversations in the training data, if the conversation with the user is already in the training data the confidence is 1.0, but if the conversation doesn't follow any story the result is 0.0.
- **Keras Policy:** As not always the user will follow the standard paths in the training data this policy uses a neural network to select the next action, so even if the previous policy gave a result of 0.0 the Keras Policy can still predict the correct action based on the mix of all the training data.
- **Mapping Policy:** Works with certain intents that only require a unique response and has no relation with the rest of the flow, whenever an intent is mapped to an action if the intent is detected the mapped action will run and after fallback to the normal predictions.
- **Form Policy:** As an extension to the Memoization Policy, this policy handles filling a set of variables with what is called Forms, if a form is detected than the form action will run until it has completed.

Custom Actions

This is where the all Sam's functionality is stored, whenever it is required to do anything extra apart from answering with a message these custom actions come in handy.
An example of this custom action is whenever a user speaks with the bot, it first has

to run the action of checking if the user actually is part of the application.
For this story:

```
## Known user
* greet
- action_check_profile
- slot{"user_exists": true}
- slot{"user_name": "Matthew"}
- slot{"measure_user": true}
- slot{"user_gender": "male"}
- slot{"user_id": "1"}
- utter_greet
```

Whenever the bot detects that the user is greeting him the first thing to do figure out who that user is, this is done with "action_check_profile" which does the following.

```
class CheckProfile(Action):
        def name(self):
                return "action_check_profile"

        def run(self, dispatcher: CollectingDispatcher, tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

                db = Database.get_instance()
                user_exists, user_id, user_name, user_gender, measure_user =
                    db.check_user_exists(tracker.sender_id)
                return [SlotSet("user_exists", user_exists),
                            SlotSet("user_name", user_name),
                                SlotSet("measure_user", measure_user),
                                SlotSet("user_gender", user_gender),
                                SlotSet("user_id", user_id)]
```

The database action queries the database for the relevant data, as you can see the action sets the corresponding sets, these are required to give Sam the context of who the user is, in the case that the user doesn't exist, Sam knows to follow a different story.

```
## Unknown user
* greet
- action_check_profile
- slot{"user_exists": false}
- slot{"user_name": null}
- slot{"measure_user": false}
- slot{"user_gender": null}
- slot{"user_id": 0}
- utter_unknown
```

The useful thing about custom actions is that it has integrated a system of reminders, this can be used to set alarms for users to measure themselves or train.

Conversation Logs

Rasa doesn't offer many logging tools but does offer the possibility of storing the Tracker information, this contains everything relevant to the conversation but mixed it shows every step the bot does while interacting with the user, in order to get conclusions and analyze the user's information a data scrubber must be implemented. This can be developed at a further date.

4.5 Testing

The way the original implementation was test-driven which means that that everything is focused around the tests.

4.5.1 Unit Tests

Tests where created for every new module of the bot before starting implementation, initially as it may seem obvious the tests failed as no code was written yet, the objective was to create tests that prove that the method was working and then create the method for it to pass the test. An example of this is the classifier test shown previously in this chapter, check section 4.4.1.

```
class TestClassifier(unittest.TestCase):

    def test_classify(self):
        tests = [
        ('hello', 'salute'),
        ('good morning', 'salute'),
        ('hi sam', 'salute'),
        ('good afternoon', 'salute'),
        ('good bye', 'bye'),
        ('bye sam', 'bye'),
        ('see you later', 'bye'),
        ('that will be all', 'bye'),
        ('I need help', 'help'),
        ('what can i do', 'help'),
        ('what commands can i say', 'help'),
        ('help', 'help'),
        ('what is the weather', 'unknown'),
        ('breakfast time', 'unknown'),
        ('where is the closest shop', 'unknown'),
        ('where can i watch the game', 'unknown'),
        ('i would like to measure myself',
        'get_measurements'),
        ('i want to measure myself', 'get_measurements'),
```

```
('i want to weigh myself',
'get_measurements'),
('new measurements', 'get_measurements'),
('configure a new table for me', 'new_table'),
('create new table', 'new_table'),
('i want a new table', 'new_table'),
('can you send me a new table?', 'new_table'),
("let's start training", 'start_training'),
("i want to train", 'start_training'),
('time to train', 'start_training'),
('start training session', 'start_training')
]

for value, expected in tests:

        with self.subTest(value=value):
                # add call to classify funtion
                result =
                    json.loads(classifier.classify({'text':
                    value}))
                intent = result['intent']
                self.assertEqual(intent, expected)
```

Until this test passed the method was tweaked. This was done with every method of the application.

4.5.2 Integration Tests

This as we explained in State of the Art section 3.8 helps the developer check if the different methods together work well together, this was going to be done for the integration of the context manager as it required several modules operating together but never was developed due to changing the design to Rasa.

4.5.3 End-to-end Tests

All end-to-end tests where done manually due to the simplicity of the bot at that stage, most tests where to verify that the webhook, the processor and the intent classifier were working well together. The plan was to create a simple bot to act as the user with predetermined conversations to verify that the bot could manage different types of conversation.

CHAPTER 5

Planning and Budget

CHAPTER 6

Socioeconomic Environment

CHAPTER 7

Conclusions

References

[1] L. Poon, “The rise and fall of new year’s fitness resolutions, in 5 charts,” 31 August, 2018. Available at https://www.citylab.com/life/2019/01/do-people-keep-new-years-resolution-fitness-weight-loss-data/579388/.

[2] T. Nwazor, “4 reasons people quit on fitness goals,” December, 2017. Available at https://www.huffpost.com/entry/4-reasons-people-quit-on_b_10396016.

[3] H. Horton, “Microsoft deletes ’teen girl’ ai after it became a hitler-loving sex robot within 24 hours,” 24 March, 2016. Available at https://www.telegraph.co.uk/technology/2016/03/24/microsofts-teen-girl-ai-turns-into-a-hitler-loving-sex-robot-wit/.

[4] E. GDPR.ORG, “Gdpr.” Available at https://eugdpr.org/.

[5] S. Worswick, “Ethics and chatbots,” 19 July, 2018. Available at https://medium.com/pandorabots-blog/ethics-and-chatbots-8d4aab75cca.

[6] C. Kumar, “Artificial intelligence: Definition, types, examples, technologies,” December, 2017. Available at https://medium.com/@chethankumargn/artificial-intelligence-definition-types-examples-technologies-962ea75c7b9b.

[7] H. Jabeen, “Stemming and lemmatization in python,” 23 October, 2018. Available at https://www.datacamp.com/community/tutorials/stemming-lemmatization-python.

[8] D. Team, “Dialogflow.” Available at https://dialogflow.com/.

[9] R. Team, “Rasa open source.” Available at https://rasa.com/.

[10] R. Polding, “Databases: Evolution and change,” 31 August, 2018. Available at https://medium.com/@rpolding/databases-evolution-and-change-29b8abe9df3e.

[11] S. Pittet, “The different types of software testing.” Available at https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing.

[12] R. Team, “Message handling.” Available at https://rasa.com/docs/rasa/user-guide/architecture/.