TÉCNICO LISBOA

Bets Helper

Bets Helper - A Sports Betting Chatbot

João Pedro Dantas Loureiro

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. José Alberto Rodrigues Pereira Sardinha

Examination Committee

Chairperson: Prof. Alberto Manuel Rodrigues da Silva
Supervisor: Prof. José Alberto Rodrigues Pereira Sardinha
Member of the Committee: Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur

May 2019

Acknowledgments

First of all, I would like to thank my parents for their love over all these years and for helping me grow to who I am today.

I would also like to thank my dissertation supervisor Prof.José Alberto Sardinha, who helped me whenever I needed and coordinated me in the best way possible and without whom this project would not have been possible.

Last but not least, I am very grateful to all my friends for the encouragement, support and for always being there for me, for putting up with me during this final project and for helping whenever i needed, including during the testing of my system. I would not have made it without you.

To you all, Thank you.

Abstract

The fast growth of Online Sports Betting around the world has led to the increase of the number of betters and, with them, the demand for betting information. In line with this demand, the objective of this project is to develop a sports betting chatbot, in other words, a sports betting personal assistant able to interact using natural language, being it via either voice or text-based conversations. The purpose of our chatbot is to collect the betting information that is spread throughout several platforms and provide it to the user on demand, in order for the users to avoid wasting time on manually searching for the information on the different platforms and so they get it as fast and with less work as possible. Also, there are no other chatbots available for the sports betting domain and no other platform supply a service similar to ours, so, in addition to enhancing the users' betting experience, it is also an innovative system for users to use. As for the technology, we created our system using Google's DialogFlow service for the chatbot logic and our own code for the information retrieval component.

Keywords

Online Sports Betting; Chatbot; Sports Betting Personal Assistant; Natural language; Betting Information; DialogFlow.

Resumo

O rápido crescimento das Apostas Desportivas Online em todo o mundo levou ao aumento do número de apostadores e, com ele, a procura por informações de apostas. Em linha com esta procura, o objetivo deste projeto é desenvolver um chatbot de apostas desportivas, ou seja, um assistente pessoal de apostas desportivas capaz de interagir utilizando língua natural, seja via voz ou texto, e com o objetivo de reunir informações de apostas espalhadas por várias plataformas e fornecê-las ao utilizador quando solicitado, para que, desta maneira, os utilizadores evitem perder tempo a procurar manualmente as informações nas diferentes plataformas e, assim, obterem-nas o mais rápido e com o menos trabalho possível. Além disso, não há outros chatbots disponíveis na área das apostas desportivas e nenhuma outra plataforma fornece um serviço semelhante ao nosso, assim, além de melhorar a experiência de apostas dos utilizadores, é também um sistema inovador para os utilizadores usarem. Quanto à tecnologia, nós criámos o nosso sistema utilizando o DialogFlow, um serviço da Google, para a lógica do chatbot e código escrito por nós para a componente de recuperação de informação.

Palavras Chave

Apostas Desportivas Online; Chatbot; Assistente Pessoal de Apostas Desportivas; Língua Natural; Informações de Apostas; DialogFlow.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Acronyms

AWS	Amazon Web Services
FAQ	Frequently Asked Questions
QA	Question Answering
SNS	Simple Notification Service

1

Introduction

Contents

In this thesis, we will discuss the creation of a sports betting chatbot which will allow users to get information regarding sports betting, including betting tips, statistic probabilities and hour and date for matches, either via voice or text-based conversations.

1.1 Motivation

Since its inception Online Sports Betting has had a steady growth but, in recent years, this growth got a lot faster and reached new heights, with an estimated global market worth of 56.7 billion dollars by 2018 [1]. Such growth and size mean that this industry has a lot of potential users for new tools that improve people's experience while betting.

Following this global fast growth is Portugal, especially after the first license for online sports betting was emitted to Betclic in May of 2016, one year after online sports betting was legalized in June, 2015 [2].

Another big booster of sports betting popularity was the release of Placard, a game released by the Departamento de Jogos da Santa Casa da Misericórdia de Lisboa in September, 2015 which is very similar to online sports betting [3]. A game from such a popular institution brought a lot of attention to sports betting which contributed to the growth of online sports betting. Placard also has had a fast growth even to this day, being the entry point to sports betting for many Portuguese people.

All factors together have led to a 122.5 million euros gross revenue from portuguese online gambling, which includes both online sports betting and online casinos, in 2017 and a 36.5 million euros gross revenue on the last trimester of the same year, a growth of almost 25 percentile points when compared to the same trimester in 2016 [4].

Figure 1.1 shows the percentile of bets for each sport in Portugal for the year of 2016 [5]. It clearly shows football is the prime candidate for the Portuguese betters when choosing a sport to bet on, making it the main focus of our project.

In order to decide in which matches to bet, betters resort to a variety of platforms that each provide a different kind of information. In each of these platforms, users have to manually search for the information they want, none have a system to offer information to users on demand. That is where our system wants to improve. Plus, as will be explained on chapter 2, bets are quite time sensitive so a system that joins the information of several different types of platforms and provides it on demand is innovative and very helpful since it saves a lot of time for betters.

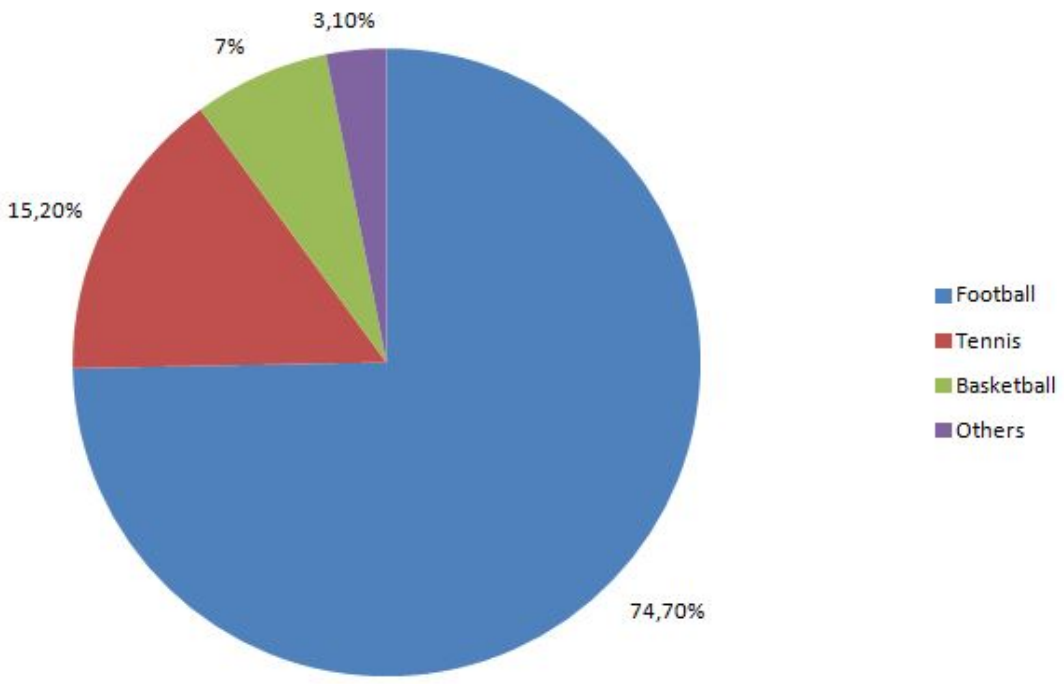

Figure 1.1: Online Sports Betting by sport

1.2 Contributions

The main objective of this project was to develop a sports betting chatbot that collects the betting information that is spread throughout several platforms and that provides it to the user on demand.

Our purpose was to create a tool that made it so users would not waste time manually searching for betting information on different platforms and so they could get it as fast and with the least work as possible.

To do so, our system is able to provide users with sports betting relevant information about:

- **Main Odds**
- **Statistic probabilities**
- **Betting tips**
- **Match Dates**
- **Team's Next Matches**

We collect this information from four different platforms so we could provide it to users through a single interface. No other existing platform offers the amount of information ours does since most platforms specialize in a single type of information.

In order for users to understand how to use the system, our chatbot has an help command. On startup, the system explains its purpose, how to use the commands to ask for information and that by simply saying ”help” it will further explain the commands.

In addition, our chatbot helps the user by steering the conversation in the right direction. It does this by helping the user identify the match they want to get information about and by remembering the match the user specified previously so users do not have to specify the same match several times in a row.

Finally, the system was made in a way users would enjoy using it and to be easy and intuitive to use. It provides the information users request in the simplest way possible.

1.3 Thesis Structure

Our thesis is divided into six chapters. In chapter 2 we explain the key concepts needed to understand the rest of the thesis. Next, in chapter 3 we present some of the most used types of platforms for sports betting and some of the existing scientific papers related to our project. After that, in chapter 4 we discuss the features, architecture and how we implemented the system. In chapter 5 we explain what we did in order to test our system and examine the results from those tests. Finally, in chapter 6 we present the conclusions of this project.

2

Background

There are a few important basic concepts when it comes to understanding betting, the most important being odds. The odds for an event reflect the probability of the outcome of that same event and, at the same time, determine the payout of a certain bet. For example, placing a bet of 1 at 2.0 odds, in case of a win, pays 2 and nothing in the case of a loss.

The two most common ways to represent odds are the decimal odds, shown as 1.22, and fractional odds, shown as 2/9, being the latter used mostly in the United Kingdom. Using fractional odds, placing a bet at, for example, 2 to 9 (2/9) odds represents a potential profit of 2 for every 9 bet. Meaning, a bet of 9 at 2/9 odds, in case of a win, pays 11 and nothing in the case of a loss.

Fractional odds are calculated using the following formula:

$$Odd = \frac{1}{EventProbability}. \tag{2.1}$$

While decimal odds are calculated by adding 1 to the previous formula, basically DECIMAL ODDS = FRACTIONAL ODDS + 1:

$$Odd = \frac{1}{EventProbability} + 1. \tag{2.2}$$

There is also the concept of bookmaker, a platform used to place bets (e.g. Bet.pt[1], Pinnacle[2]). Through bookmakers you can bet on the outcome of a certain event in a game for the offered odd. Bookmakers offer odds according to the probability they think is accurate for a certain outcome, which is why they change their offered odds over time and according to the circumstances.

[1] www.bet.pt
[2] www.pinnacle.com

Crystal Palace - Liverpool

Saturday, 31 Mar 2018, 12:30

1X2 | Home/Away | AH | O/U | DNB | EH | More bets

Full Time | 1st Half | 2nd Half

Bookmakers	1	X	2	Payout
10Bet	6.80	4.75	1.47	96.4%
18bet	7.00	4.80	1.48	97.4%
1xBet	6.80	5.00	1.50	98.6%
5Dimes	7.40	4.91	1.47	98.1%
Asianodds	6.94	4.99	1.48	98.0%
bet-at-home	6.16	4.58	1.47	94.3%
bet365	7.00	4.33	1.44	93.6%
Bethard	6.80	4.80	1.45	95.7%
BetOlimp	6.46	4.79	1.51	97.5%
Betrally	6.70	4.70	1.45	95.1%
bwin	7.00	4.75	1.44	95.4%
Marathonbet	6.90	5.00	1.49	98.4%
Pinnacle	6.94	4.99	1.48	98.0%
TonyBet	7.00	4.95	1.45	96.7%
Unibet	7.00	4.95	1.47	97.5%
Vulkanbet	6.06	4.51	1.46	93.3%
Click to show 58 more bookmakers!				
= Average	6.81	4.80	1.47	96.5%
Highest	7.40	5.00	1.51	100.3%

Figure 2.1: Oddsportal list of odds for Crystal Palace - Liverpool match

Different bookmakers offer different odds, even for the same event outcome, which is why most betters use several bookmakers. If a bookmaker offers higher odds when compared to others, for the same event, betters may choose them in order to maximize profits.

Odds comparison platforms allow users to compare the odds provided by different bookmakers for the same match, making it easier for users to choose a bookmaker. Figure 2.1 shows one of those platforms, Oddsportal[3]. Here we can see the list of odds provided by different bookmakers for the Crystal Palace vs Liverpool match.

[3] www.oddsportal.com

On the left side, we have the list of bookmakers, followed by three columns of the odds of a home win, a draw and an away win, respectively. For example, 5Dimes is the bookmaker that offers the highest odds of Crystal Palace winning the match, while BetOlimp offers the highest odd of Liverpool winning.

The payout column is the sum of these three columns, the difference from 100 represents the profit for the bookmaker. Also represented are both the average and highest of each column, below the list.

The final concepts are sports tips, tipsters and tipster platforms. Sports tips are suggested bets for the likely outcome of a sports event. These are provided by tipsters through tipster platforms (e.g. Blogabet[4], Bettingexpert[5]), which are mostly websites where the tipsters, the people who come up with the tips, publish and share them for other people to follow, either with or without a match analysis as well.

As explained before, odds can vary over time. Actually, it is very common for odds to change because of new information about a game, to better suit the event or simply for bookmakers to minimize their losses. Adding to this, if too many people follow a tip and start betting on the same match, the tendency is for the odd to drop so bookmakers do not lose money. This makes tips quite time sensitive and consequently, the earlier a better follows a tip, the better the profit.

[4]www.blogabet.com
[5]www.bettingexpert.com

3

Related Work

Contents

In this chapter we examine the most used types of platforms when it comes to sports betting in order to see what they have to offer and how they compare to our system. We also delve into the existing relevant scientific research related to our project.

3.1 Tipster Platforms

Tipster platforms allow people to publish and share tips with others. These tips can, sometimes, include an analysis so people understand the though process behind the tip.

The people that publish these tips are normally experts and are called Tipsters. Some tipsters require payment in order for people to have access to their tips, normally a monthly fee, while others do it for free, usually to win competitions paid by the tipster platform.

Among the platforms, most offer the ability to follow a certain tipsters, but only a few allow users to receive notifications regarding new tips.

In this section we examine some of the most widely used tipster platforms, compare the features each one has to offer and explain our choice on which one to get the information from.

3.1.1 Platforms

Academia das Apostas [1] - *Academia das Apostas* is a Portuguese website that includes analysis from the editors for some of the most popular matches, articles about betting and betting news, competitions for tipsters to win money prizes and a forum for users to discuss tips and other betting related subjects. It also allows users to follow tipsters so newly inserted tips appear as notifications on the website. Notifications are not sent by email so the users are required to visit the website to view them.

Bet Advisor [2] - *Bet Advisor* has a blog updated by the editors with articles about betting, sport news and match analysis for a few important games. They allow both free and paid tipsters. In order to have access to paid tipster tips, users are required to either pay a monthly subscription and have access to all the tips or buy tips individually and only have access to the tips bought. It is not possible to follow free tipsters, only paid tipsters. Notifications are received in real time but are also a paid service, included in the monthly subscription to a tipster. Below, Figure 3.1 shows some of the active tips a tipster name Alexander Johnson had for the day of 18/05/2018.

[1] www.academiadasapostas.com
[2] www.betadvisor.com

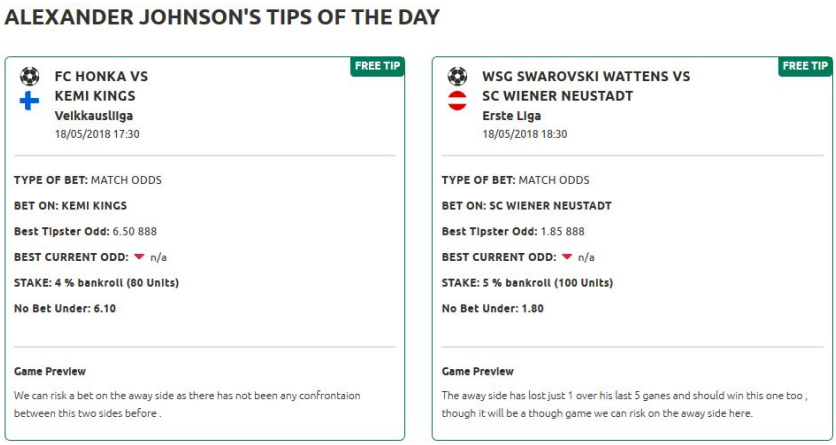

Figure 3.1: Bet Advisor - Example of a tipster's active tips

Betshoot [3] - *Betshoot* provides analysis from the editors for a few games while also offering a section where users can comment on this analysis. They run a monthly competition with money prizes for the top tipsters but do not allow users to subscribe to tipsters.

Bettingexpert [4] - *Bettingexpert* runs a monthly competition for tipsters with a money prize for the best ones. They allow users to subscribe to tipsters and receive notifications when new tips are added. Figure 3.2 shows three tips from three different tipsters for the Marseille vs Atletico Madrid match. The first tip suggests betting on Atletico for the fulltime win, the second suggests betting on Atletico for the fulltime win with an handicap and the third suggests betting on Marseille for the fulltime win with and handicap. Also shown is the tip summary of all the tips posted for that same game.

[3] www.betshoot.com
[4] www.bettingexpert.com

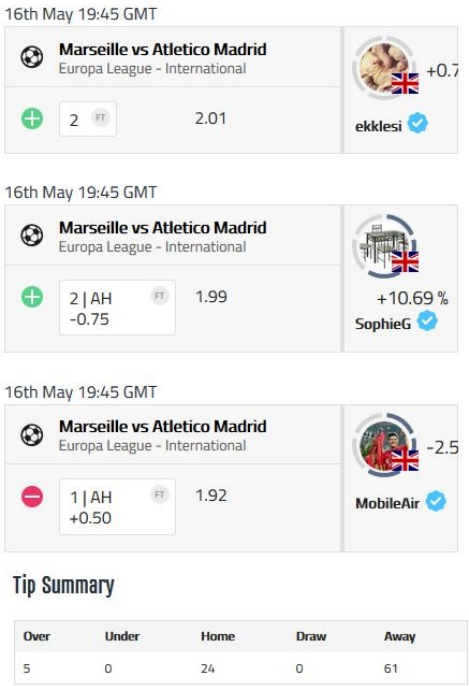

Figure 3.2: Bettingexpert - Some tips and tip summary for the Marseille vs Atletico Madrid match

Blogabet [5] - *Blogabet* does not have competitions for tipsters. They allow both free and paid tipsters and it's possible to follow both categories. Paid tipsters are allowed to post some free tips but in order to have access to all tips, users need to pay a monthly subscription. Notifications by both email and phone are available, but are paid services, otherwise notifications require the user to check the website.

OLBG [6] - *OLBG* features a forum divided by sport where users can discuss betting subjects related to that sport. They run a wide range of monthly tip competitions for several categories and for specific competitions (e.g. Football World Cup). It is possible to follow tipsters and receive notifications when new tips are inserted via email.

3.1.2 Comparison

To help in comparing all the previously mentioned platforms, we created the table seen on Figure 3.3 which contains the features of each platform.

[5] www.blogabet.com
[6] www.olbg.com

	Academia das Apostas	Betshoot	OLBG	Bettingexpert	Blogabet	Bet Advisor
Paid Tipsters	✗	✗	✗	✗	✓	✓
Articles	✓	✗	✗	✗	✗	✓
Editor match analysis	✓	✓	✗	✗	✗	✓
Forum	✓	✗	✓	✗	✗	✗
Competitions	✓	✓	✓	✓	✗	✗
Tipster Following	✓	✗	✓	✓	✓	✓***
External Notifications*	✗	-	✓	✗	✓**	✓****
Real-time notifications	-	-	✗	-	✗	✓

Figure 3.3: Comparison table for the tipster platforms

When we look at the table we can see that articles and match analysis are not widely offered by these platforms, being available on two and three platforms respectively. These can be very useful features but are very hard to implement through a chatbot because of the amount of information we need to convey, so we are not going to focus on these.

Also, we can see that only three of the platforms provide external notifications, as in email or text message, to users that follow tipsters. Of those three, only OLBG[6] offers this as a free service, the other two require payment. Also, Bet Advisor[2] is the only platform that offers real-time notifications, which are included on their paid tipster subscriptions.

Finally, we can see tipster platforms are divided into two different models when it comes to rewarding tipsters. Either they have competitions that reward the best tipsters on the platform or they allow tipsters to be paid for their tips.

So, if users are willing to pay, Bet Advisor[2] is the best platform. It allows users to follow tipster, provides real-time notifications and also has articles and match analysis.

However, if users are looking for a free service, OLBG[6] is the best. It is the only platform that provides external notifications, although not real-time, for free. It lacks the articles and match analysis but those can be found on Academia das Apostas[1]. Those two platforms compliment each other to provide the

* - Email or phone notifications
** - Paid Service
*** - Only to paid tipsters
**** - Paid service included on the monthly subscription to a tipster

best experience for free users.

On our project we offer two different types of tip information: (a) a tip summary of all the tips posted for a specific match; and (b) tips from a specified tipster for a specific match.

To collect this information we relied on webscraping, which is explain on Section 4.4. We choose to scrape this information from Bettingexpert[4] because it was the only platform which offered a tip summary, simplifying our job, and also because it was one of the easiest platforms to scrape information from.

The main difference from our project is that, even tho we offer similar information, none of these platforms offer the information to users on demand, the user has to manually search for the relevant information. Our chatbot improves on this model by allowing users to get the information they want via voice or text request, without the need to specifically look for it. It is also innovative in the sports betting area since none of the existing platforms do anything similar.

3.2 Match Prediction Platforms

Besides tipster platforms, another popular service among betters is match prediction platforms. These platforms make predictions for the outcome of matches based on statistical data, which may include previous match results of both teams, previous direct confrontations between the two teams, injured players, among others.

Since this prediction is purely mathematical it may not be a hundred percent accurate, but it can still be useful when it comes to help betters determine the best bet.

In this section we are going to present two such platforms and examine the features they have to offer.

3.2.1 ProSoccer

ProSoccer[7] is a website which offers free football predictions. Their predictions are generated on a weekly basis by a computational intelligence software, using artificial neural networks and sports data entered into their system.

This technology allows the probabilities of the possible outcomes for the match to be deduced, which allows users to see what the machine ”thinks” logically about the game without any human bias.

The available predictions include:

- The probability of the full time win[8] of the home team, draw and win of the away team;
- The probability of the game ending with over and under 2.5 goals;

[7] www.prosoccer.gr
[8] A full time win refers to a win at the end of a match, while an half time win is given to the team that is winning at the half-time

- The two most likely final scores;

Also displayed is the tip for each match, which is the suggested bet for the full time outcome, and the average odds for the the full time win of the home team, draw and win of the away team.

Figure 3.4 shows a real example of the predictions for four different matches. For each match, we can see the league, the teams playing, the predicted probability for each of the three possible outcomes at the end of the match, the tip, the average odds for each outcome, the most likely and second most likely predicted scores, the probabilities of both teams scoring under and over 2.5 goals during the match and, when the match is over, the final score.

League	Match	Predict. Prob% 1	Predict. Prob% X	Predict. Prob% 2	Tips	Avg. Odds 1	Avg. Odds X	Avg. Odds 2	Pred. Score 1	Pred. Score 2	Under 2.5 Goals	Over 2.5 Goals	Final Score
ENF	CHELSEA - MANCHESTER UNITED	72	20	8	1	2.80	3.00	2.50	3-0	2-0	41	59	-
ENF	MANCHESTER UNITED - CHELSEA	31	28	41	2 1	2.45	3.10	2.75	0-1	0-2	50	50	-
ES1	VILLARREAL - REAL MADRID	24	28	48	2 X	2.60	3.75	2.25	0-2	0-1	45	55	-
DEC	BAYERN MUNCHEN - EINTRACHT FRANK	75	20	5	1	1.20	5.85	10.4	4-0	2-0	43	57	-

Figure 3.4: ProSoccer Matchs Example

3.2.2 Statarea

Statarea[9] also is a website that offers free football predictions. Here, however, there is also user discussion and feedback.

As on ProSoccer[7], a tip for each match is offered to users, but here users can give their feedback in the form of a like or dislike. Users can also comment on the match predictions and even give their own prediction for the full time result.

The predictions offered by Statarea[9] are more extensive, they include:

- The probability of the full time win[8] of the home team, draw and win of the away team;
- The probability of the half time win[8] of the home team, draw and win of the away team;
- The probability of the game ending with over 1.5, 2.5 and 3.5 goals;
- The probability of both teams scoring and only one team scoring;

Figure 3.5 below shows an example for the Marseille vs Atletico Madrid. Here we can see the tip, followed by the users feedback, the predicted probability for the three possible outcomes at the end of the

[9]www.statarea.com

match and at the half-time, the probabilities of the game ending with over 1.5, 2.5 and 3.5 goals scored, the probability of both teams scoring and of only one team scoring and, finally, the users' prediction for the match.

Figure 3.5: Statarea Match Example

3.2.3 Comparison

Both platforms offer their prediction services for free. ProSoccer[7] has the advantage of using a computational intelligence software to generate their predictions while Statarea[3] does not specify how they generate theirs, but Statarea[9] offers a wider variety of predictions and allows users to give their opinion and feedback.

Our project improves upon these platforms by offering the same available information on demand, via voice or text request, so users do not have to manually search the website for the desired information.

To collect this information we relied on webscraping, which is explain on Section 4.4. We could have scraped the information from any of the two websites but in the end we chose Statarea because of the wider selection of available predictions and also because it was easier to scrape than the alternative.

3.3 Scientific Research

In this section we delve into the existing relevant scientific research related to our project.

3.3.1 Chatbots: Are they Really Useful?

In this paper [6] Shawar & Atwell investigate areas where chatbots could be useful, such as education, information retrieval, business and e-commerce. They also present several chatbots with useful applications found in daily life, such as help desk tools.

They start by explaining that a chatbot system is a software program that interacts with users using natural language. The authors also give a brief historical introduction to chatbots and explain how the technology evolved, starting when it began in the 1960's with the aim of mimicking human conversation and amuse the user.

A main focus of the paper is the ALICE chatbot system which stores its knowledge using AIML, Artificial Intelligence Mark-up Language. The authors explain AIML's format and how it works, and ALICE's pattern matching algorithm.

The authors then present a Java program they developed that converts a text corpus to the AIML in order to re-train ALICE. This program had two versions: (a) the first one which is based on simple pattern template category matching; and (b) the second one that uses a machine-learning category generation approach.

In the rest of the paper, the authors talk about using chatbots for four different purposes:

As a Tool of Entertainment - When talking about chatbots as tool of entertainment, the authors explain that initial aim of building chatbot systems was to mimic human conversation and amuse users. They give the example of ELIZA [7], the earliest example of a chatbot in this paper and which was created in the 60's by Joseph Weizenbaum to emulate a psychotherapist in clinical treatment. It used keyword matching, meaning that the input was inspected for a keyword and the answer formulated according to a rule associated with that keyword. If the keyword was not found, a connected free remark was used.

As a Tool to Learn and Practice a Language - In order to see if chatbots could help users learn and practice a language the authors started by using the second version of the program they developed. This program allowed them to automatically build AIML from a corpus, even if they did not understand the domain or even the language. This made it possible for them to read a Corpus of Spoken Afrikaans, convert it to the AIML format files and then re-train ALICE to create two Afrikaans chatbots.

The authors then mounted prototypes of the chatbots on websites using Pandorabot service and encouraged open-ended testing and feedback from remote users in South Africa. During these tests, users found that many responses were not related to the topic or were nonsense but they still found the chatbot an interesting tool to practise the language and enjoyed chatting with it.

After analysing similar experiments [8,9], the authors found that chatbots could be useful for students learning new languages or practicing old ones for several reasons, including: (a) Chatbots being able to repeat the same material with students several times without being bored; (b) Many chatbots using both text and speech mode when responding, which provides an opportunity to practice reading and listening skills; and (c) Chatbots being a new trend, which improves the students motivation towards learning.

As an Information Retrieval Tool - The authors give several examples of chatbots used as an information retrieval tool. The leading example being FAQChat, which was a retrained version of ALICE after the author's program was adapted to the FAQ (Frequently Asked Questions) of the School of Computing at the University of Leeds. FAQChat provides answers about topics that include the Unix operating system, teaching and research resources.

The other examples include: (a) YPA [10], which is a chatbot that allows users to retrieve information from British Telecom's Yellow pages. The yellow pages contain advertisements, with the advertiser

name and contact information, so it answers users' questions by returning addresses and, if no address is found, a conversation is started and the system asks users more details in order to give the user the required address. YPA could answer questions such as "I need a plumber with an emergency service?" and "Which restaurants are there in Colchester high school?"; and (b) Virtual Patient bot (VPbot) [11], which simulates a patient that medical students can interview. It was successful in Harvard Medical School where students who used it scored higher marks on exams.

Chatbots in Business, E-Commerce and other Fields - Finally, the authors give three examples of chatbots in other domains: (a) There is Happy Assistant [12], a chatbot that helps users access e-commerce sites to find relevant information about products and services; (b) Sanelma, a chatbot that impersonates a fictional person that users can talk with in a museum and that provides background information concerning a certain piece of art; and (c) Rita (real time Internet technical assistant) [13], used in the ABN AMRO Bank to help customers do financial tasks such as money transfers.

After having surveyed several chatbot systems which succeed in practical domains like education, information retrieval, business, e-commerce, as well as for amusement, the authors conclude that in general, the aim of chatbot designers should be to build tools that help people, facilitate their work and their interaction with computers using natural language, but not to replace the human role totally, or imitate human conversation perfectly.

We found this paper relevant for our project first because it gave us an historical introduction to chatbots, it showed us how they worked and what they were used for since the creation of the technology. It also showed us that chatbots can be successful in several domains, including information retrieval. This is a positive sign for our project because with the growing popularity and more widespread use of chatbots in several areas, a chatbot for sports betting should be well received among users. Also, our chatbot works basically as a information retrieval chatbot since it collects information from several sources when the user requests, so this paper gave us an assurance that it would be a viable solution.

Finally, we also share the authors' opinion, our chatbot is built as a tool to help users and facilitate their work but not to replace the human component, users still have their role that the chatbot can not replace.

3.3.2 Case Study: Building a Serverless Messenger Chatbot

In this paper [14] Lehvä, Mäkitalo & Mikkonen present a two-month case study on building and designing a Facebook Messenger chatbot, for a media company, that could help the users follow up on the latest news of their interests. The goal of the project was to design a scalable, modern architecture for a chatbot that follows liquid software principles [15].

The authors start by introducing and explaining the technologies they used to create the chatbot: (a) Facebook Messenger; and (b) Amazon Web Services.

The interface for the chatbot is provided by the Facebook Messenger, a service for instant messaging with Facebook friends. To start a conversation with the chatbot, the user simply has to search for the chatbot by the name of the Facebook page they are linked to on the Messenger.

Facebook does not offer any solution to build the chatbot backend, so the backend needed to be built and hosted somewhere else. To connect with the backend, the platform offers callbacks which call a webhook when a user sends a message so it gets delivered to the backend. After processing the question, the chatbot sends the answer to the Facebook Send API which then delivers it to the user as a reply to the initial message.

To build the backend, the authors used the Amazon Web Services (AWS), which is a cloud service platform built by Amazon the offers a wide range of services for developing software. More specifically, from the services offered by AWS they used:

Lambda - Lambda is a Serverless environment where the code is uploaded to and that can then be triggered by events from Mobile apps, HTTP endpoints and other AWS services. This is where all the chatbot logic is stored but it is not explained what techniques they used to create the chatbot.

Simple Notification Service (SNS) - SNS is a web service that can be used to send notifications from the cloud.

CloudWatch - CloudWatch is a monitoring system for AWS cloud resources. It provides easy way to read logs, set up alarms, monitor system performance and resource utilization and set up web dashboards to make visualizations of all those together. CloudWatch also supports making scheduled events to invoke other AWS resources such as the SNS or Lambda. The authors use it to invoke a Lambda function once every minute to read news from a third-party API.

API Gateway - API Gateway is used as a gateway for incoming requests for other AWS resources, such as Lambda. In this case, when a user sends a message using Messenger, it triggers a callback to a webhook that contains URL pointing to the API Gateway which then passes the event to a Lambda function.

DynamoDB - DynamoDB is a NoSQL database that supports both document and key-value store models.

Next, the authors explain the motivation behind using a serverless computing approach and why they decided to use a serverless architecture, implemented using AWS, for their project.

Using a serverless computing approach, the developer does not need to manage servers. Instead, the developer can simply just upload the code to the serverless environment and the code gets executed when it is triggered by some event. Also, the serverless environment does the scaling automatically depending on the workload.

All this means the developer is free from worrying about scaling, security updates and any other work related to keeping the environment up and running, as all that work is outsourced to the service

provider [16–18]. Furthermore, serverless computing can be very cost efficient as the customer only pays when the code is being executed and also because, due to not requiring system administration work, the serverless approach can also lead to cost savings in operational management [16].

Finally, the authors report their findings regarding the design and the final implementation of their system and compare these findings to other recent studies about using a serverless approach to build a chatbot.

Similarly to the other studies [16, 18–20], they found that the serverless approach is, in most cases, very inexpensive and requires close to zero effort to manage since it there is no need to manage servers and the scaling is done automatically.

Also, some of the studies said that serverless services can be hard to debug [16, 20]. Likewise, their case study also had signs of the debugging problems but they were mostly solved by using the existing tools more efficiently. The reason why that could have been an issue for the other studies could be explained by different, older or not so developed platforms.

We found this paper relevant for our project because it shares some important similarities to ours. The most important being:

- Both chatbots are used for information retrieval, albeit for different themes: sports betting on our case and news providing for theirs. This gave us an example of how these chatbots work which helped us get a better understanding for building ours.
- Both rely on a serverless approach, even tho with different providers: they used AWS while we rely on services provided by Google. This paper gave us a very good insight on both the advantages and disadvantages of using a serverless approach in order to build a chatbot, since the authors go in depth into the analysis of this theme.

In conclusion, this paper gave us a real example of what we were trying to accomplish, a information retrieval chatbot using a serverless approach, gave us an idea of the difficulties we could face, as well as possible solutions to these problems, and, more importantly, showed us that it would be possible.

3.3.3 Existing systems

In this section we analyse two smaller papers that present existing systems that can be compared to our project and from which we learned important matters that we later used for our project, not for the chatbot part in itself but for other important parts, such as the system evaluation.

3.3.3.A YourQA

In the first paper [21], Quarteroni & Manandhar report their experience on the design, implementation and evaluation of a chatbot-based dialogue interface for an open-domain question answering (QA) sys-

tem, while showing that chatbots can be effective in supporting interactive QA.

The system is named YourQA and it is able to provide both factoid and complex answers such as definitions and descriptions. It takes the top 20 Google results for a question, retrieves the corresponding Web pages and analyzes them to extract answers and rank them by relevance to the question.

The authors based their system on an AIML interpreter. They used Chatterbean3, a Java-based AIML interpreter, which allows them to define custom AIML tags and allows a seamless integration between the QA module and the chat interface. The interactive interface consists of an applet window with a left panel where the chat takes place and a right panel where results are visualized when read.

A non-interactive interface already existed for the system where users could enter a question in a text field and obtain a list of answers in the form of an HTML result page.

Dialogue interfaces pose complex evaluation challenges and so are often evaluated using qualitative metrics such as user satisfaction and perceived time of usage [22].

So, when it came to evaluate their system, the authors designed three scenarios where users had to look for two different items of information relating to the same topic. Users first used the non-interactive Web interface of the QA prototype and then the interactive version to find answers.

After that, users were asked to fill in a questionnaire about their experience with the interactive version and about which version they preferred. It was composed of nine questions where question one and two assessed the performance of the system, question three and four focused on interaction difficulties, five and six related to the overall satisfaction of the user, seven and eight focused on the system performance and question nine assessed which of the versions the users preferred.

In the end, the authors drew optimistic conclusions on the feasibility of chatbot based interactive QA. This is relevant for our project because it is another example of a successful information retrieval chatbot we learned from.

Most importantly, this paper was relevant because we used it to learn how to evaluate our system. It explained the difficulties faced when evaluating a system like this and gave us an understanding on the options we had to evaluate our system, including introducing us to the user evaluation approach, which they used and that we ended up using as well. Also, it provided us with additional material we learned from for this same matter.

3.3.3.B System for providing real time sports betting information

In the second paper and in a different vein from all the previous papers, we have a patent [23] for a method and system for providing real time sports betting information and where Corbo explains the motivation behind building a system for online sports betting and describes the system in question, including examples of several different screens of the system.

The system is built to provide sports betting information to users in real time, in order to allow users

to make more intelligent decisions regarding their bets. Its features include:

- Providing the user with a schedule of all sporting events for the given time period, and the different bets that can be made on each game. These events and bets each have their own corresponding odds, which are given to the user for all of the different sports books that a bet can be placed at.
- Providing the user with a means to log and track bets for a complete total of winnings and losses. This bet tracker feature, as it is called, allows the user to log each and every bet he makes into the system, which in turn calculates and lists a total of winnings and losses.
- Alerting the user when a predetermined occurrence takes place in connection with the odds. These alerts include one for a predetermined line becoming available and another for a “major line move” between the various sports books and casinos. For example, if the user wishes to place a bet at specific odds, he can set the line seeker alert, and that will notify him when any sports book offers those particular odds. The major line move alert can be set to notify the user if a predetermined amount of sports books change their lines, on a specific game, within a predetermined amount of time.

Regarding our project, this paper allows us to compare our project to an existing sports betting information retrieval system, and even though it is not a chatbot, it still a relevant system in the same domain as our own. It also gave us an idea of which sports betting information might be useful for users when it comes to making the right decisions for their bets.

4

Proposed Solution

Contents

In this chapter we will go into further detail on the features and implementation of the system. We will also describe the chosen architecture, meaning we will explain each part that constitutes the system, why we chose them and how they work together.

4.1 Functionality

Our system is first and foremost a personal assistant or, in other words, a chatbot. That means it is capable of interacting with users using natural language, either via voice or text-based conversations. In this case, the chatbot works in English.

Being a sports betting oriented chatbot means it provides information that helps users chose the right bets to make. We built our system to provide five different types of information:

Main Odds - Users can ask for the odds for a match, in which case the system returns the main odds for the indicated match, which include odds for: (a) the full-time result[1]; (b) the half-time result[2]; (c) over and under 0.5, 1.5 and 2.5 goals; (d) both teams to score or not; (e) double chance[3]; and (f) draw no bet[4].

Statistic probabilities - Users can ask for statistically predicted probabilities, in that case the system returns probabilities for: (a) the full-time result[1]; (b) the half-time result[2]; (c) over 1.5, 2.5 and 3.5 goals; and (d) both teams to score. Beyond these, the system also returns a tip based on this probabilities.

Betting tips - Users can ask for betting tips for a given match in either of two ways: (a) If they specify a tipster, the system returns that tipster's tip for that match; or (b) if they do not, the system returns a tip summary made from all the tips tipsters posted for the specified match.

Date - Users can ask for the date of a match between two teams, in this instance the system returns the date for the next time the two specified teams play.

Team Next Matches - Users can ask for the next matches of a team, in which case the system returns the next five matches of the specified team, in chronological order.

The chatbot is made to be user friendly, so we added several features in for it to better steer the conversation in the right direction and to help the users get the information they want in the easiest way possible:

1. When the user inputs only a match with no information request, the chatbot asks the user what type of information they would like to know about that match.

[1] A full-time result can be either home team win, away team win or draw

[2] A half-time result refers to the result at a match's half-time which can be a draw, home team winning or away team winning

[3] A double chance is a bet that consists of betting on two possible outcomes, meaning either home team win or draw, away team or draw and either team to win

[4] A draw no bet refers to betting in a team winning but the bet money is returned if the game ends in a draw

2. When a user requests information about a match, after answering that request, the chatbot asks if the user would like to know anything else about that match. This way the user does not have to input the same match several times in a row.

3. When the chatbot can not find the match a user requested information about, it provides several sugestions of matchs that the user might be talking about. These come in the form of each team's next match.

4. The chatbot has a help command with several options. When asked for help in general, the chatbot explains it's purpose, what commands are available and how to use them to get information. When asked for help for a specific command, the chatbot explains what that command does and how to use it.

4.2 Architecture Overview

To help better understand the architecture of our system we created a simple diagram, as can be seen in Figure 4.1. It shows our system is divided intro three parts:

Google Assistant - Which acts as the user interface. Google Assistant receives the user input and passes it as text to Dialogflow. The opposite happens for the output, the response is received as text and converted to speech for the user.

Dialogflow - Which contains the chatbot logic. Dialogflow parses the text input in order to understand what the user wants and passes this information to Google Cloud Functions. After receiving the response, Dialogflow passes it to Google Assistant.

Google Cloud Functions - Which hosts the code of the system. It receives the request from Dialogflow, scrapes the needed information and returns the final processed response to Dialogflow.

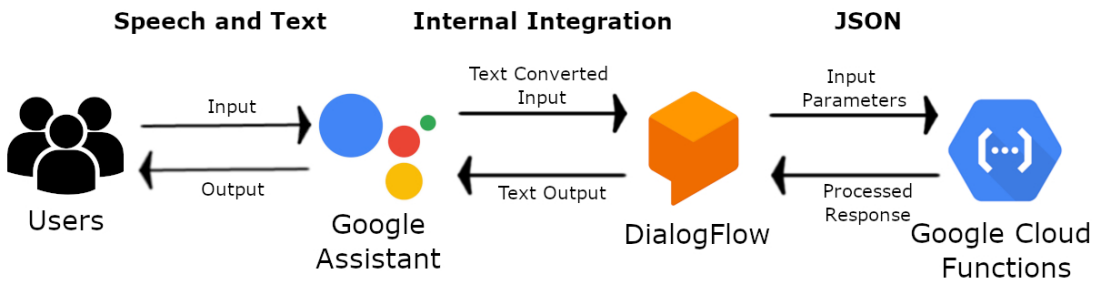

Figure 4.1: System Architecture

The first step for building our system was the creation of the Dialogflow agent, followed by writing, hosting and connecting the code and finally integrating Google Assistant with Dialogflow. Each of these parts is going to be further individually explained in the following sections.

4.3 Dialogflow

Dialogflow is a service owned by Google that makes it easier for users to create a chatbot by providing a platform that allows developers to create agents, which house all the logic needed for the human-computer interaction, while incorporating Google's machine learning expertise [24]. A new agent comes pre-programmed with basic greetings, fallbacks and small talk, which can then be customized and further expanded to allow conversations for the desired effect.

We found Dialogflow was, by far, the easiest and lest time-consuming way to create a chatbot since it saved us the work of creating all the logic needed to handle the human-computer interaction. We started by creating a Dialogflow agent, BetsHelper.

The base of each agent are Intents. These represent the different subjects the agent is supposed to be able to discuss.

We created Intents for each of the features the agent has. Figure 4.2 shows the Intents menu with all the Intents we use, each one for a specific purpose:

Date - Answers user's queries about the date of a specific match;

Fallback Intent - As the name suggests, works as a fallback. It is the Intent the agent uses when it can not match the input phrase to any Intent;

Welcome Intent - Triggered at the start of the conversation to welcome the user;

Help - Answers the user's requests for help;

Match - Handles the case where the user gives a match but does not specific what they want to know about that match;

NextMatches - Triggered when the user asks for a specific team's next matches;

Odds - Answers the user's queries about odds for a specific match;

Statistics - Handles the case where the user asks for stats for a certain match;

Teamcont - Triggered when the user wants to know additional information about the match from the previous query;

Tip - Answers the user's queries about tips for a specific match;

User Tip - Handles the case where the user asks for a specific tipster's tip for a certain match;

Yes - Used to answer certain user inputs in order to help the conversation move forward.

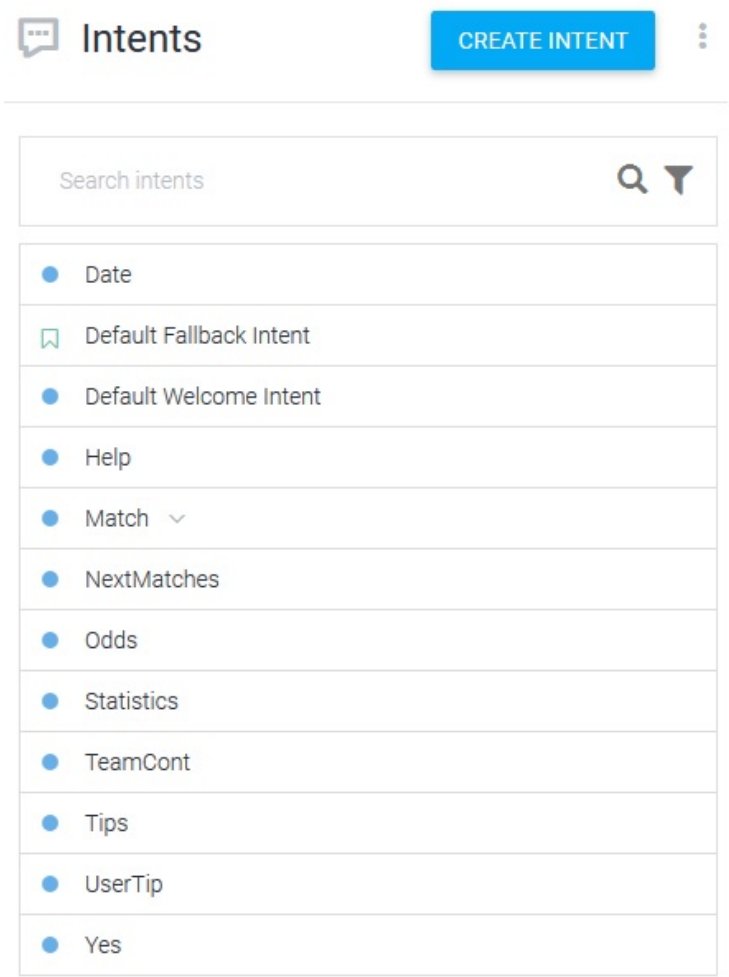

Figure 4.2: Dialogflow Intents Menu

Within each of these Intents we specified several Training Phrases, which are examples of possible input phrases specified by the developer. These are needed so the agent can determine the Intent each input phrase belongs to. It does this by determining the closest match between the input phrase and the Training Phrases of all Intents.

We specified around twenty phrases depending on the Intent. For the simpler Intents we used less phrases while, for the more complex intents, we used a larger collection of phrases since there were more ways for the users to talk.

Figure 4.3 shows some of the Training Phrases for the Tips Intent. In this case, the Training Phrases are the different ways the user can request tips for a match.

The agent also needs to know what information is useful within each input phrase. To do this, developers specify Entities which are key concepts needed to answer the user's request. Dialogflow includes a few predefined system entities, like the date Entity for instance.

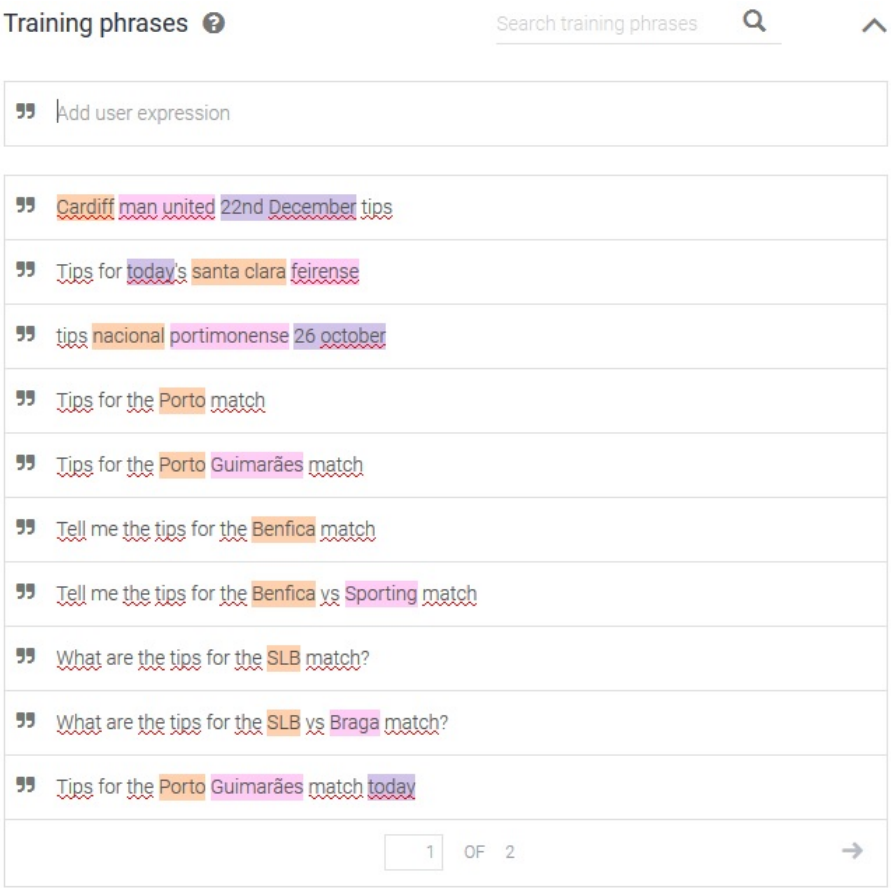

Figure 4.3: Training phrases from the Tips Intent

We use a total of four, although they vary accordingly to the intent. We use one system Entity, the date one, and three created by us, these being the Decision, Subject and team Entities, as can be seen on the Entities menu shown on Figure 4.4.

Figure 4.4: Entities Menu

The Decision Entity is simply yes or no, the Subject Entity represents the feature the input phrase is requesting to use, it can be ”Odds” or ”Stats” for instance, and the team Entity represents the English Premier League teams. Figure 4.5 shows one training phrase from the Odds Intent. We can see highlighted the recognized Entities in the training phrase, these being two separate team Entities and one date Entity.

Figure 4.5: One training phrase from the Odds Intent

The last part we had to set up on Dialogflow was the Fulfillment. While the Intents identify what the user wants, the Fulfillment is the code that is hosted elsewhere, connected to Dialogflow through a webhook and that handles the response.

When Dialogflow receives an user input, it matches it to an Intent and sends an HTTP POST request to the webhook with a JSON object containing the information about the Intent and the Entities of that same input. The answer also comes in the form of a JSON object containing the answer ready to be delivered to the user. The code and what it does is explained in Section 4.4.

The second big advantage of using Dialogflow, after not having to create the chatbot logic, is the ease of training the agent. Using Google’s machine learning technology, an agent learns both from the examples provided in the training phrases and from the language models developed by Dialogflow. Based on this data, it builds a model for making decisions on which intent should be triggered by a user input and what data needs to be extracted. It then keeps improving though real conversations with users.

It is also possible to view the past conversations logs and if necessary manually assign certain inputs to a specific intent. We had to do this in very few occasions in order to fine tune the agent.

4.4 Code

The code we wrote for our system has the main purpose of formulating a response for the user query using the information retrieved via webscraping. We choose to write our code in the Python programming language mainly because of Beautiful Soup [25], a Python library for pulling data out of HTML and XML files which makes webscraping a lot simpler and easier. We had to resort to webscraping, which is the act of extracting data from websites, because there are no available APIs that provide the information

we require.

When Dialogflow sends the request, the code handles it and extracts the Intent and the Entity parameters.

It then checks the Intent in order to know what information the user wants since each Intent relates to a feature, as explained in Section 4.3. Regardless of which Intent is used, the Entity parameters derived from the user's input are used as a base to webscrap the information.

Take the example request shown on Figure 4.6, the input phrase is shown on the "queryResult", "queryText" section: "Give me the odds for the man city arsenal on 03/02". Dialogflow matched this phrase with the Odds Intent, as shown on the "intent", "displayName" section and found three Entities which it sent as parameters, as can be seen on the "queryResult", "parameters" section: (a) "team1" matched to "man city" which results in the "Manchester City" parameter; (b) "team2" matched to "arsenal" which results in the "Arsenal" parameter; and (c) "date" matched to "03/02" which results in the "2019-02-03T12:00:00+00:00" parameter.

Diagnostic info

RAW API RESPONSE | FULFILLMENT REQUEST | FULFILLMENT RESPONSE | FULFILLMENT STATUS


```
{
  "responseId": "d1beced0-c2ba-4a7a-96fe-996b930127ae",
  "queryResult": {
    "queryText": "Give me the odds for the man city arsenal on 03/02",
    "parameters": {
      "team1": "Manchester City",
      "team2": "Arsenal",
      "date": "2019-02-03T12:00:00+00:00"
    },
    "allRequiredParamsPresent": true,
    "intent": {
      "name": "projects/sports-mockup/agent/intents/1fca3d89-2dd0-4a7b-8b5d-07657664659f",
      "displayName": "Odds"
    },
    "intentDetectionConfidence": 0.84,
    "languageCode": "en"
  },
  "originalDetectIntentRequest": {
    "payload": {}
  },
  "session": "projects/sports-mockup/agent/sessions/8a030133-8068-71ab-b142-2f178ade68f9"
}
```


Figure 4.6: Example Fulfillment Request

For each Intent our code does the following:

Date - Given two teams, gets the date of their next match;

Help - Finds out what type of help the user needs;

NextMatches - Finds the next five matches of the given team;

Odds - Given two teams and a date, webscrapes the odds for that match;

Statistics - Given two teams and a date, finds the statistics for that match;

Tip - Finds the tip summary when given two teams and a date;

User Tip - Given two teams, a date and a tipster, webscrapes that tipster's tip for the match;

Finally, using the acquired information, we formulate the response the chatbot will give the user and send it to Dialogflow.

For a more in-depth view, the full code can be found on Appendix A at Listing A.1.

4.5 Google Cloud Functions

We chose to host our code using Google's Google Cloud Functions service, which is an event-driven serverless compute platform part of Google Cloud Platform.

We opted for a serverless approach because it allows us to simply deploy our code to Google. Google Cloud Functions runs the code and dynamically scales it to match the usage. This shifts the worry about server management and security patches away from the developer [16–18, 26, 27].

We chose to use Google Cloud Functions because being an serverless platform, it enjoys the benefits explained previously but also because it provided a webhook which we needed to set up the Dialogflow Fulfillment. It was the easiest and least time consuming way of setting up the code hosting and the Fulfillment part of our project.

Google Cloud Functions also provides logging and monitoring for our code. Figure 4.8 shows us our function control panel which allows us to edit and test our function but also shows our function monitoring. Both this and the logs, which Figure 4.7 shows some examples of, are very important features because they allow us to debug our function and track its performance.

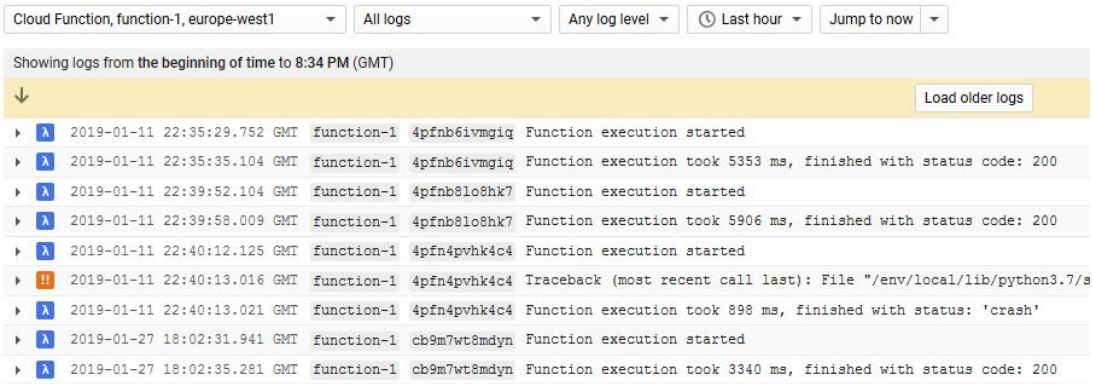

Figure 4.7: Google Cloud Functions Logs

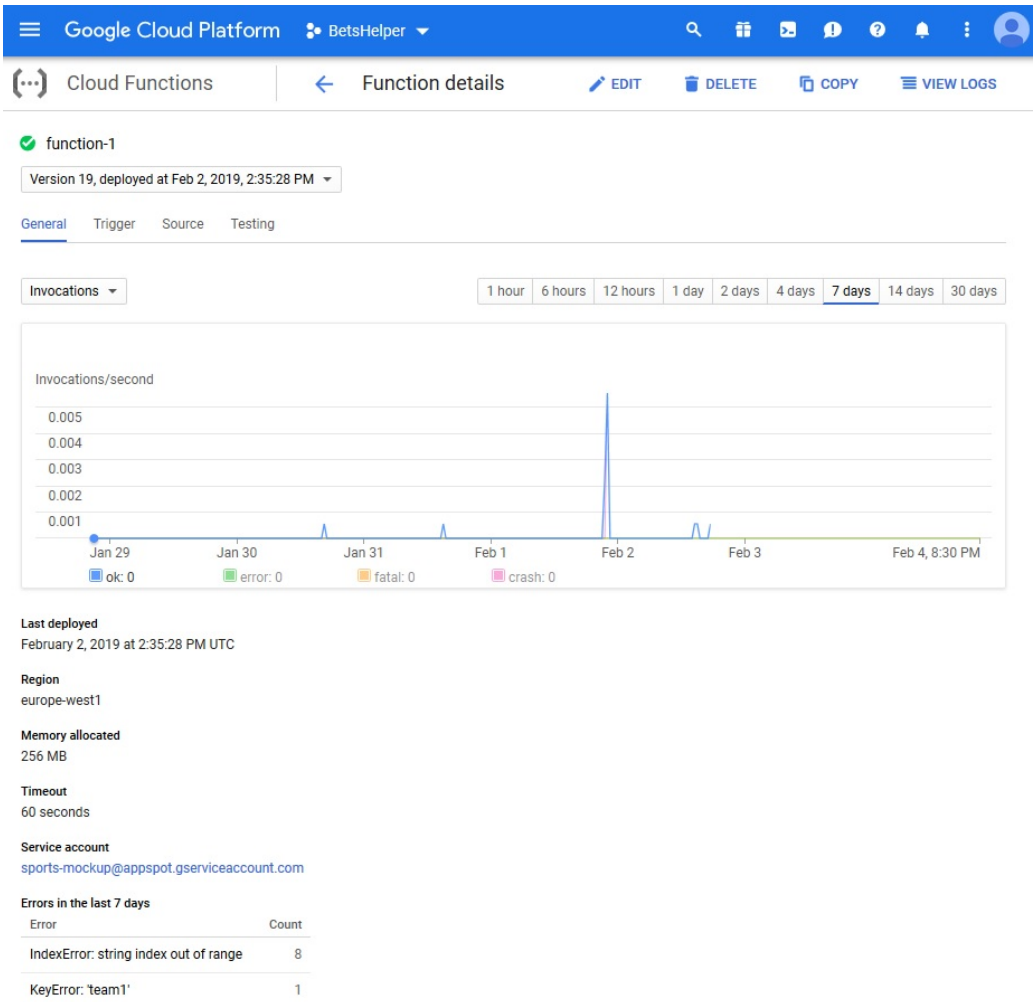

Figure 4.8: Google Cloud Functions Function

4.6 Google Assistant

Google Assistant is a virtual assistant developed by Google, in other words, it is a software agent that can perform tasks or services for an individual. It is able to communicate with users via both voice and text, meaning it interprets human speech and responds via synthesized voice.

We chose this platform for the user interface. This means Google Assistant is the visual part of our system through which a user interacts. It receives the user input and passes it as text to Dialogflow while the opposite happens for the output.

We decided to use Google Assistant as the user interface instead of developing a new one because the integration between Dialogflow and Google Assistant is immediate. Within Dialogflow we have the option to export the agent to Google Assistant, requiring just a few other additions, which include device selection and submission for approval from Google, for the agent to be operational through it. Also it

is customizable and, since it is owned by Google, we can expect good functionality and high design standards.

This decision saved us the time and work of building a user interface from scratch and is the much easiest and least time consuming way when compared to the available alternatives.

Figure 4.9 below is an example of our Google Assistant interface. It shows an interaction where the user asks for the next matches being played by Manchester City.

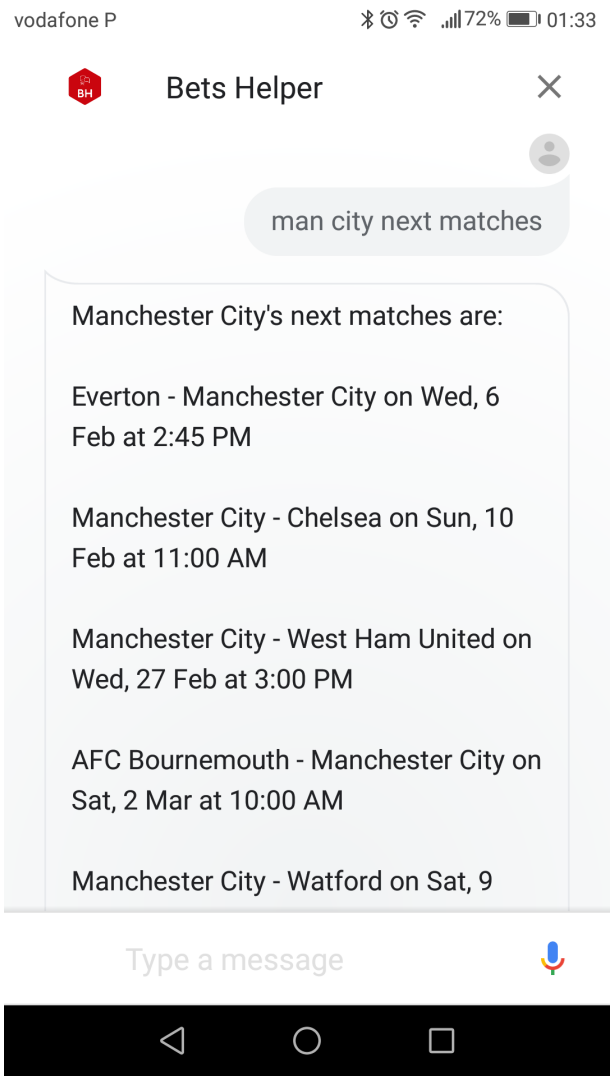

Figure 4.9: Google Assistant Example

5

Work Evaluation

Contents

In order to evaluate our system, since dialogue interfaces pose complex evaluation challenges, we decided to test its performance, usability and user satisfaction through the users test evaluation approach [28]. This approach consists of creating several tasks that encompass the overall usage of the system, having the testers do them and examining how they perform. This allows us to register metrics such as the time of usage which we use to evaluate the system's usability. Following the tests, we also had the testers fill a questionnaire in order to measure their satisfaction with the system.

In this chapter we explain what the testing phase of our project consisted of and examine the results of these same tests.

5.1 Testing

For the user testing we started by developing the use cases, which are tasks that encompass all the functionalities and that reflect the intended typical usage of the system. For this purpose we came up with the following four tasks:

T1 - Find Tottenham next matches and the date for the Chelsea vs Tottenham match;

T2 - Get the probabilities for the Chelsea vs Tottenham match;

T3 - Get the odds for the Chelsea vs Tottenham match;

T4 - Get the tips for the Chelsea vs Tottenham match and the tip by "MisterTips" for that same match.

These tasks were subjected to pilot tests in order to guarantee that they were well formulated and that made sense to testers. Also, each task was also given a clear definition of when it was successfully completed so we could accurately record the usability metrics. For example, the third task would be successfully completed as soon as the chatbot presented the odds.

Next we choose our usability metrics, which were the metrics we recorded in order to evaluate the system. We settled for the following three:

Task Duration Time - The time the tester took to complete the task;

Number of Errors - The number of errors the tester did during the task, with an error being a mistake that required the task to be restarted from the starting point;

Task Completeness - If the tester was capable of finishing the task with success.

With both the tasks and the metrics defined, we then proceeded on to the actual testing. We started by giving a short introduction with less then five minutes which explained to the testers what is a chatbot and how to communicate with our system, what is sports betting and what kind information is possible to obtain from our system. After this explanation the users were allowed to use the system for 2 to 3 minutes.

Following this introduction, we moved on to the tasks. As stated before, the users were asked to complete four tasks while we recorded the time they took and the number of errors committed. Each

tester did the four tasks twice, first using our system and then using the existing alternatives. This was done so we could compare both metrics and conclude which platform is best for the users.

Since no other platform offers all the services our system offers, different tasks had to be compared to different platforms. So task 1 was compared to Google[1], task 2 to Statarea[2], task 3 to Betclic[3] and task 4 to Bettingexpert[4].

We collected the times from the tests with thirty different users and calculated the average and standard deviation of the time they took to complete each task. The following Table 5.1 shows both this times, the total number of errors the users made in each task and if they were able to complete the task or not.

	Average Time		**Errors**		**Completed With Success**	
Task	**Bets Helper**	**Alternatives**	**Bets Helper**	**Alternatives**	**Yes**	**No**
1	19.29±2.47	32.06±6.09	0	0	30	0
2	11.51±3.02	35.90±23.05	0	8	30	0
3	12.83±4.26	19.46±4.70	1	0	30	0
4	20.50±10.75	40.79±19.22	3	7	30	0

Table 5.1: Tasks Result Table

As can be seen from the results, our system is faster in all tasks, with the alternatives taking, in some cases, more than double the time of our system to provide the same information. Also, the testers were more prone to mistakes using the alternatives than when using our system.

The errors that occurred when using our system due to the user inputting mistaken information (the wrong match date, for example). As for the alternatives, both the platforms used during the second and fourth task, Statarea[2] and Bettingexpert[4] respectively, can be hard to understand if never used before, which caused the larger number of errors.

Also, all users were able to finish all four tasks with success.

To better examine the results, we also calculated averages dividing the results between experienced and inexperienced betters. As is better explained in the next Section 5.2, out of the thirty testers, twenty-one were considered inexperienced betters while the remaining nine were considered experienced betters.

The following Tables 5.2 and 5.3 show both these averages and the errors.

	Average Time		**Errors**	
Task	**Bets Helper**	**Alternatives**	**Bets Helper**	**Alternatives**
1	18.96±1.52	24.78±2.33	0	0
2	10.09±2.51	27.40±25.41	0	1
3	13.49±9.03	14.74±0.47	1	0
4	18.81±6.30	40.07±33.09	1	3

[1] www.google.com
[2] www.statarea.com
[3] www.betclic.pt
[4] www.bettingexpert.com

Table 5.2: Experienced Better Tasks Result Table

	Average Time		Errors	
Task	**Bets Helper**	**Alternatives**	**Bets Helper**	**Alternatives**
1	19.44±2.82	35.18±3.05	0	0
2	12.11±3.16	39.55±22.30	0	7
3	12.55±1.73	21.48±3.59	0	0
4	21.23±12.33	41.10±14.73	2	4

Table 5.3: Inexperienced Better Tasks Result Table

From these results we can see that our system is faster for both groups but the inexperienced betters are considerably faster using our system. Also, inexperienced betters are more prone to mistakes than experienced betters when using the alternatives.

Our system seems like an improvement over the alternatives for all users but mostly for inexperienced betters, mostly because it is easier to get information by speaking than by using platforms never used before and because our system is made to be user friendly.

Finally, we created graphics (figs. 5.1 to 5.4) with the times each tester took for each task, using both our system and the corresponding alternative.

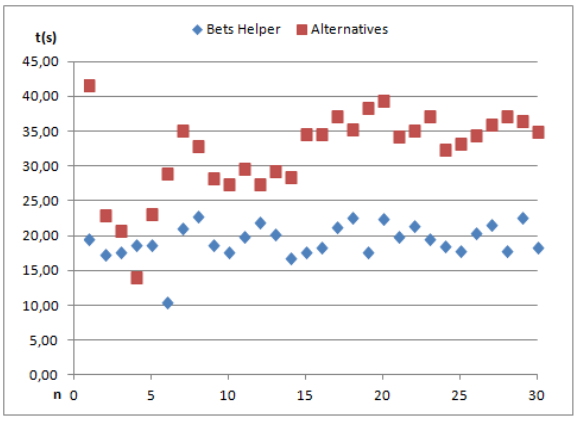

Figure 5.1: Task One Results Graph

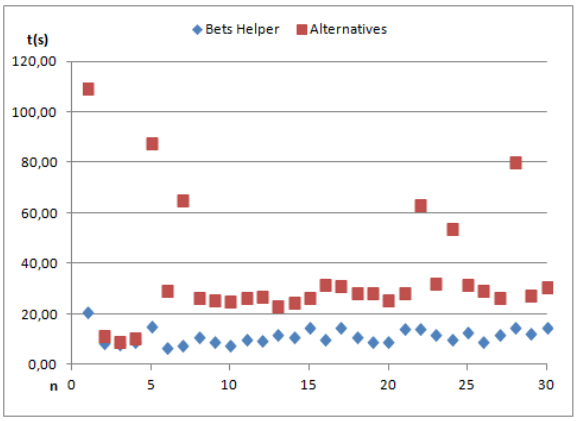

Figure 5.2: Task Two Results Graph

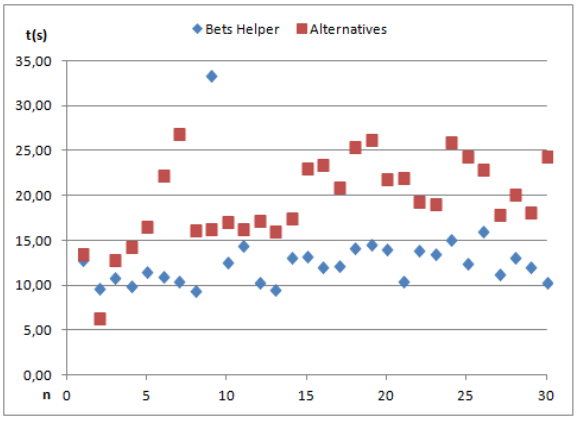

Figure 5.3: Task Three Results Graph

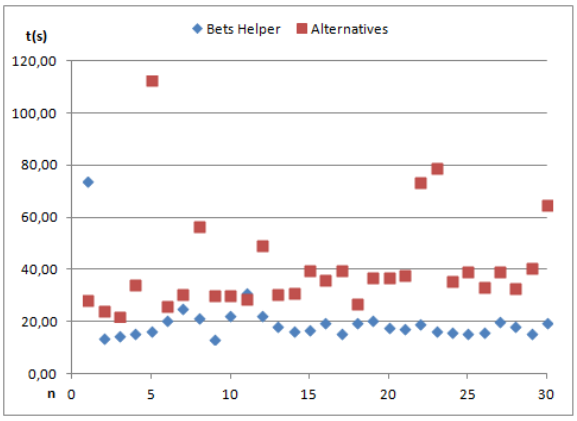

Figure 5.4: Task Four Results Graph

The horizontal axis represents the testers, each number is a person, and the vertical axis represents the time in seconds. Each tester has two times, the one in blue for the time it took to complete the task using our system and the one in red for the time it took to complete the task using the corresponding alternative for that task.

These graphics help us get a better idea of the results using a visual representation and emphasise the results the conclusions we reached from the tables. We can clearly see that our system is faster except for a few very rare exceptions.

5.2 Questionnaire

After finishing the tasks, each tester was asked to fill a user satisfaction questionnaire (see appendix B). The questionnaire was comprised of eight questions and was designed to be simple and fast to answer, as not to over saturate the testers and so the tests did not take too much of their time. The questions were divided into two parts, the first one was the user characterization part, where we collected information about the testers, and the second part where the testers gave their opinion on the system's usability.

As mentioned previously, we did tests with thirty different people. We collected and processed the answers each one gave in order to create tablestables 5.4 to 5.7 so we could better analyse the results.

The first three questions of the questionnaire were multiple choice questions and aimed to characterize the tester. On Tables 5.4 and 5.5 we can see the total count of the different answers the testers gave for this three first questions. We can see the majority of the testers were male with an age between 21 and 27 years but we also tried to get testers outside this demography.

1. Sex		**2. Age**						
Male	**Female**	**<14**	**14-17**	**18-21**	**21-27**	**28-35**	**36-41**	**>41**
24	6	0	0	2	20	3	2	3

Table 5.4: Testers Sex and Age Table

3. Academic Qualifications				
Basic Education	**Secondary Education**	**Degree**	**Masters Degree**	**PhD**
1	17	9	3	0

Table 5.5: Testers Academic Qualifications Table

We tried to get a diversity of testers from different backgrounds because our system is supposed to be used by a large demography which involves many different kinds of people, so our tests aimed to encompass as many different people as possible.

The following two questions were meant to assess the sports and betting knowledge of the testers. They were answered using a scale from 1="Never" to 5="Every Day", where users had to choose one as the answer.

Table 5.6 shows the count of answers for this questions. From this table we can see that most testers have a fairly decent following when it comes to sports, with twenty-five of the answers being either 3 or 4, and that one third of the testers have a good knowledge when it comes to betting, with answers 3 or above, which we considered experienced betters, while the other two thirds had very little knowledge of sports betting with answers of 2 or below, which we inexperienced betters.

	1	**2**	**3**	**4**	**5**
4. How often do you follow sports?	0	3	13	12	2
5. How often do you engage in sports betting?	11	10	4	5	0

Table 5.6: Testers Sports Engagement Table

Our system is built to be user friendly and to be used by both new and experienced betting users so it was important to test it with both kinds of user in order for us to compare it to the alternatives and to identify the different difficulties each group had.

The last three questions assess the testers opinion on the system which is essential for us to examine the system usability and user satisfaction. Questions 6 and 7 were both answered using a scale from 1 to 5, where on question 6 1="Very Hard" and 5="Very Easy" and on question 7 1="Not at all" and 5="Very". Question 8 was a multiple choice question. Table 5.7 shows us the collected answers for this questions.

	1	**2**	**3**	**4**	**5**	**Average**
6. How easy was it to obtain the desired information?	0	0	1	21	8	4.23±0.5
7. Did you enjoy using the system?	0	0	0	23	7	4.23±0.43

	Bets Helper	**Alternatives**
8. When comparing Bets Helper to the alternatives, which one was your favorite?	30	0

Table 5.7: Testers Satisfaction Table

From the table we can see the count of answers, with questions 6 and 7 also having the average with standard deviation. We can see the answers are very positive, with 29 and 30 of all testers rating 4 or above on both questions 6 and 7, which translates to an answer average of 4.23 for both questions.

Question 6 was meant to assess the system usability while question 7 was meant to assess the user satisfaction with the system, so this kind of feedback is a very positive sign for our system since it means that the system is good at supplying the desired information and that the big majority of users like using it.

Also, question 8 shows that every tester preferred using our system to the alternatives. We should note that some experienced betters stated that, while they preferred using our system for these specific tasks, for normal betting some of the alternatives may be have to be used occasionally for specific situations. But they also stated that having access to all the different kinds of information our system offers is very useful and saves a lot of time when betting that would be otherwise spent alternating between platforms.

The feedback from the testers was majorly positive, with the only complaints being: (a) the language since some users would have preferred to use the system in Portuguese, which is something not currently possible mostly because Google Assistant does not work in Portuguese; and (b) the very few times where the user asked for information in a way the chatbot did not understand, in which case we manually taught the chatbot after the fact.

6

Conclusion

In recent years, online sports betting has had a very fast growth and reached a new peak in popularity. We took advantage of this growing industry to create a system that would improve the betting experience for both the new and experienced betters by gathering some of the most important betting information scattered through several platforms and by providing it through a chatbot, which is an innovative technology in the field.

We used a Google platform named Dialogflow to create the chatbot. It took some time to learn how to use it but in the end we managed to create a chatbot with great results. In order to get the information needed for our system to work we used our own written code, which in the end worked in a fast and efficient manner. Using a combination of these two elements we managed to build a good and very solid system.

The biggest obstacle we encountered while building the system was the problem of feeding the information we got with our code to the Dialogflow platform for two main reasons. First, it took some time to understand how Dialogflow communicated with the exterior, and secondly, we needed to find a solution to host our code. For Dialogflow, we simply needed to set up a webhook and adapt our code to communicate through JSON. For the hosting, after exploring the solutions, we ended up settling for Google Cloud Platform since it is a serverless platform and provides a number of features we benefited from.

After having the system working, we needed to test and evaluate it. To do so we opted for the users test evaluation approach with thirty users, using tests that reflect the real world usage of the system. The results we collected from these tests were very positive, the system performed better than the alternatives, all while providing several types of information that previously required four different platforms to be consulted.

Also, the system was made to be user friendly and the results reflected that. The users easily adapted to the system, found it easy to use and enjoyed using it. The use of the chatbot technology was a good choice in this matter since it is a type of technology easy for users to adapt to. It was especially good for users with very small betting experience because it is more intuitive than the alternatives, which required much more experience to use effectively.

Overall, from this results, we can say we were successful in both implementing what we proposed to and in creating a good system for betters to use.

Bibliography

[1] Gambling 911. ”How Big is the Online Sports Betting Industry?”. Accessed 02-February-2019. [Online]. Available: http://www.gambling911.com/gambling/how-big-online-sports-betting-industry-.html

[2] Diário de Notícias. ”Já se aposta online em Portugal. ”É a liberdade”, dizem jogadores”. Accessed 02-February-2019. [Online]. Available: https://www.dn.pt/sociedade/interior/ja-se-aposta-online-em-portugal-e-a-liberdade-dizem-jogadores-5193711.html

[3] Observador. ”Santa Casa lançou novo jogo de apostas desportivas”. Accessed 02-February-2019. [Online]. Available: https://observador.pt/2015/09/08/santa-casa-lancou-novo-jogo-de-apostas-desportivas/

[4] Diário de Notícias. ”Jogos e apostas online geraram 122,5 milhões de euros em 2017”. Accessed 02-February-2019. [Online]. Available: https://www.dn.pt/portugal/interior/jogos-e-apostas-online-geraram-1225-milhoes-de-euros-em-2017---relatorio-9129319.html

[5] Expresso. ”Quanto é que apostas?”. Accessed 02-February-2019. [Online]. Available: http://expresso.sapo.pt/sociedade/2017-07-01-Quanto-e-que-apostas-#gs.jhcKBsY

[6] B. A. Shawar and E. Atwell, “Chatbots: Are they really useful?” *LDV Forum*, vol. 22, no. 1, pp. 29–49, 2007. [Online]. Available: http://www.jlcl.org/2007_Heft1/Bayan_Abu-Shawar_and_Eric_Atwell.pdf

[7] J. Weizenbaum, “Eliza—a computer program for the study of natural language communication between man and machine,” *Commun. ACM*, vol. 9, no. 1, pp. 36–45, Jan. 1966. [Online]. Available: http://doi.acm.org/10.1145/365153.365168

[8] J. Jia, “Csiec (computer simulator in educational communication): An intelligent web-based teaching system for foreign language learning.”

[9] L. Fryer, “Bots as language learning tools,” *Language, Learning and Technology*, vol. 10, 01 2006.

[10] U. Kruschwitz, A. De Roeck, P. Scott, S. Steel, R. Turner, and N. Webb, “Natural language access to yellow pages,” in *Proc. (Cat. No.99TH8410) 1999 Third Int Conf. Knowledge-Based Intelligent Information Engineering Systems*, Aug. 1999, pp. 34–37.

[11] G. M. Weber, “Data representation and algorithms for biomedical informatics applications,” Ph.D. dissertation, Cambridge, MA, USA, 2005, aAI3174069.

[12] J. Chai, V. Horvath, N. Nicolov, M. Stys-budzikowska, A. Kambhatla, and W. Zadrozny, “Natural language sales assistant - a web-based dialog system for online sales,” in *Proc. 13th Innovative Applications of Artificial Intelligence Conference, IAAI’01*, 2001, p. pp.

[13] D. Voth, “Practical agents help out [virtual agent system],” *IEEE Intelligent Systems*, vol. 20, no. 2, pp. 4–6, Mar. 2005.

[14] J. Lehvä, N. Mäkitalo, and T. Mikkonen, “Case study: Building a serverless messenger chatbot,” in *Current Trends in Web Engineering - ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, practi-O-web, NLPIT, SoWeMine, Rome, Italy, June 5-8, 2017, Revised Selected Papers*, ser. Lecture Notes in Computer Science, I. Garrigós and M. Wimmer, Eds., vol. 10544. Springer, 2017, pp. 75–86. [Online]. Available: https://doi.org/10.1007/978-3-319-74433-9_6

[15] A. Taivalsaari, T. Mikkonen, and K. Systä, “Liquid software manifesto: The era of multiple device ownership and its implications for software architecture,” in *Proc. IEEE 38th Annual Computer Software and Applications Conf*, Jul. 2014, pp. 338–343.

[16] Mike Roberts. Serverless Architectures. Accessed 02-February-2019. [Online]. Available: https://martinfowler.com/articles/serverless.html

[17] A. Eivy, “Be wary of the economics of ”serverless” cloud computing,” *IEEE Cloud Computing*, vol. 4, no. 2, pp. 6–12, Mar. 2017.

[18] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, “Serverless computation with openlambda,” in *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*. Denver, CO: USENIX Association, 2016. [Online]. Available: https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/hendrickson

[19] M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang, “Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures,” in *Proc. Cloud and Grid Computing (CCGrid) 2016 16th IEEE/ACM Int. Symp. Cluster*, May 2016, pp. 179–182.

[20] M. Yan, P. Castro, P. Cheng, and V. Ishakian, “Building a chatbot with serverless computing,” in *Proceedings of the 1st International Workshop on Mashups of Things and APIs*, ser. MOTA ’16. New York, NY, USA: ACM, 2016, pp. 5:1–5:4. [Online]. Available: http://doi.acm.org/10.1145/3007203.3007217

[21] S. Quarteroni and S. Manandhar, “A chatbot-based interactive question answering system,” *Decalog 2007*, vol. 83, 2007.

[22] M. Walker, C. Kamm, and D. Litman, “Towards developing general models of usability with paradise,” *Natural Language Engineering*, vol. 6, no. 3-4, pp. 363–377, 2000.

[23] D. Corbo, “Method and system for providing real time sports betting information,” Nov. 15 2011, uS Patent 8,057,300.

[24] Dialogflow. Dialogflow. Accessed 02-February-2019. [Online]. Available: https://dialogflow.com/

[25] Beautiful Soup. Beautiful Soup Documentation. Accessed 02-February-2019. [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[26] Google. Serverless Computing — Google Cloud. Accessed 02-February-2019. [Online]. Available: https://cloud.google.com/serverless/

[27] ——. Google Cloud Functions - Event-driven serverless compute platform — Google Cloud. Accessed 02-February-2019. [Online]. Available: https://cloud.google.com/functions/

[28] M. J. Fonseca, P. Campos, and D. Gonçalves, “Introdução ao design de interfaces,” *FCA Editora*, 2012.

A

Code of Project

The code we wrote for our project in its integrity. It handles the request from Dialogflow, webscrapes the required information, creates the answer for the user and sends it back to Dialogflow.

Listing A.1: Project Code

```
import urllib.request
from urllib.request import Request
from bs4 import BeautifulSoup

import os
import re
import random

from flask import Flask
from flask import request
from flask import make_response
from flask import jsonify

TEAMS = {
    "Manchester City": "382",
    "Liverpool": "364",
    "Tottenham": "367",
    "Chelsea": "363",
    "Arsenal": "359",
    "Manchester United": "360",
    "Wolves": "380",
    "Everton": "368",
    "West Ham": "371",
    "Watford": "395",
    "Bournemouth": "349",
    "Leicester": "375",
```

```
    "Brighton": "331",
    "Newcastle": "361",
    "Crystal Palace": "384",
    "Cardiff": "347",
    "Huddersfield": "335",
    "Southampton": "376",
    "Burnley": "379",
    "Fulham": "370"
}

MATCHPHRASES = [
    "Would you like to know anything else about this match?",
    "Can I help you with anything else regarding this match?",
    "Is there anything else you would like to know about this match?"
]

HELPPHRASES = [
    "What else can I help you with?",
    "What else can I help with?",
    "Anything else I can help you with?",
    "Can I help you with anything else?",
    "Is there anything else I can help with?"
]

CONTPHRASES = [
    "What can I help you with regarding that match?",
    "What would you like to know about that match?"
]

def loadSoup(url):
    page = urllib.request.urlopen(url)

    return BeautifulSoup(page, 'html.parser')

def loadSoupTips(url):
    user_agent = 'Mozilla/5.0 (iPhone; CPU iPhone OS 5_0 like Mac OS X)
        AppleWebKit/534.46'
    page = urllib.request.urlopen(Request(url, data=None,
        headers={'User-Agent': user_agent}))

    return BeautifulSoup(page, 'html.parser')

def webhook(request):
    req = request.get_json()
    resp = processRequest(req)
    r = make_response(jsonify(resp))
    return r

def getParams(params):
    team1 = params["team1"]
    team2 = params["team2"]
    date = params["date"]

    return team1, team2, date

def getParamsContainer(contexts):
    for context in contexts:
        params = context["parameters"]
        if "team1" in context["parameters"]:
            break

    return params

def createContextResp(speech, contextName, team1, team2, date):

    resp = createResp(speech)

    resp = addContext(resp, contextName, team1, team2, date)

    return resp

def createResp(speech):

    resp = {
        "fulfillmentText": speech,
        "fulfillmentMessages": [
          {
```

```
            "simpleResponses": {
              "simpleResponses": [
                {
                  "textToSpeech": speech,
                  "displayText": speech
                }
              ]
            }
          }
        ]
    }

    return resp

def addContext(resp, contextName, team1, team2, date):
    resp["outputContexts"] = [
      {
        "name": contextName,
        "lifespanCount": 1,
        "parameters": {
          "date": date,
          "date.original": date,
          "team1.original": team1,
          "team1": team1,
          "team2": team2,
          "team2.original": team2
        }
      }
    ]

    return resp

def getContext(contextString):

    contextName = ""

    contextList = contextString.split("/")

    for t in contextList[:-1]:
        contextName += t + "/"

    return contextName

def randomFromList(phraseList):

    return random.choice(phraseList)

def processRequest(req):
    intent = req["queryResult"]["intent"]["displayName"]
    params = req["queryResult"]["parameters"]

    if intent == "Odds":
        team1, team2, date = getParams(params)
        speech = processOdds(team1, team2, date) + "\n  \n" +
            randomFromList(MATCHPHRASES)
        resp = createResp(speech)

    elif intent == "Statistics":
        team1, team2, date = getParams(params)
        speech = processStats(team1, team2, date) + "\n  \n" +
            randomFromList(MATCHPHRASES)
        resp = createResp(speech)

    elif intent == "Tips":
        team1, team2, date = getParams(params)
        speech = processTips(team1, team2, date) + "\n  \n" +
            randomFromList(MATCHPHRASES)
        resp = createResp(speech)

    elif intent == "UserTip":
        team1, team2, date = getParams(params)
        tipster = params["tipster"]
        speech = processSpecificTip(team1, team2, date, tipster) + "\n  \n"
            + randomFromList(MATCHPHRASES)
        resp = createResp(speech)
```

```
    elif intent == "Date":
        team1 = params["team1"]
        team2 = params["team2"]
        speech = processDate(team1, team2) + "\n  \n" +
            randomFromList(MATCHPHRASES)
        resp = createResp(speech)

    elif intent == "Help":
        subject = params["Subject"]
        speech = processHelp(subject) + "\n  \n" +
            randomFromList(HELPPHRASES)
        resp = createResp(speech)

    elif intent == "NextMatches":
        speech = processNextMatches(params["team"]) +
            randomFromList(HELPPHRASES)
        resp = createResp(speech)

    elif intent == "Match":
        speech = randomFromList(CONTPHRASES)
        resp = createResp(speech)

    elif intent == "Match - General":
        resp = processMatchGeneral(req)

    elif intent == "Match - General - Tipster":
        params = getParamsContainer(req["queryResult"]["outputContexts"])
        team1, team2, date = getParams(params)
        tipster = params["tipster"]

        speech = processSpecificTip(team1, team2, date, tipster) + "\n  \n"
            + randomFromList(MATCHPHRASES)
        contextName =
            getContext(req["queryResult"]["outputContexts"][0]["name"]) +
            "teamcont"

        resp = createContextResp(speech, contextName, team1, team2, date)

    elif intent == "TeamCont":
        if params["Subject"] != "":
            resp = processMatchGeneral(req)
        else:
            speech = randomFromList(CONTPHRASES)

            contextName = req["queryResult"]["outputContexts"][0]["name"]
            params = getParamsContainer(req["queryResult"]["outputContexts"])
            team1, team2, date = getParams(params)

            resp = createContextResp(speech, contextName, team1, team2, date)

    elif intent == "TeamContNo":
        speech = randomFromList(HELPPHRASES)
        resp = createResp(speech)

    else:
        print("else")
        speech = "Impossible."

    return resp

def processOdds(team1, team2, date):
    matchLink = getMatchLink(team1, team2, date)

    if matchLink == False:
        altMatchLink = getMatchLink(team2, team1, date)

        if altMatchLink == False:
            return "Odds not available for this match. " +
                teamsNextMatch(team1, team2)

        matchLink = altMatchLink

        team1, team2 = switchTeam(team1, team2)

    oddsList = getOddsList(matchLink)

    speech = "The odds for this match are:\n  \n"
```

```
    speech += "Fulltime: " + oddsList[0] + " for " + team1 + ", " + oddsList[2] + " for " + team2 + " and " + oddsList[1] + " for the draw.\n  \n"
    speech += "Halftime: " + oddsList[3] + " for " + team1 + ", " + oddsList[5] + " for " + team2 + " and " + oddsList[4] + " for the draw.\n  \n"
    if oddsList[6] != False:
        speech += "Over/Under goals: " + oddsList[6] + " and " + oddsList[7] + " for over and under 0.5 goals; " + oddsList[8] + " and " + oddsList[9] + " for 1.5 goals; " + oddsList[10] + " and " + oddsList[11] + " for 2.5 goals.\n  \n"
    if oddsList[12] != False:
        speech += "Both teams to score: " + oddsList[12] + " yes and " + oddsList[13] + " no.\n  \n"
    if oddsList[14] != False:
        speech += "Double chance: " + oddsList[14] + " for " + team1 + " or draw, " + oddsList[15] + " for " + team1 + " or " + team2 + " and " + oddsList[16] + " for " + team2 + " or draw.\n  \n"
    if oddsList[17] != False:
        speech += "Draw no bet: " + oddsList[17] + " for " + team1 + " and " + oddsList[18] + " for " + team2 + "."

    return speech

def getMatchLink(team1, team2, date):
    soup = loadSoup('https://www.betclic.pt/futebol/inglaterra-premier-lg-e3')

    dateString = date[0:10]
    if dateString[-5] == "0":
        dateString = dateString[0:5] +  dateString[6:]
    if dateString[-2] == "0":
        dateString = dateString[0:7] +  dateString[8]

    matchString = team1 + " - " + team2

    dateContainer = soup.find(class_="entry day-entry grid-9 nm", attrs={"data-date": dateString})

    if dateContainer is None:
        return False

    gameContainer = dateContainer.find(class_="match-entry clearfix CompetitionEvtSpe ", attrs={"data-track-event-name": matchString})

    if gameContainer is None:
        return False

    gameContainer = gameContainer.find(class_="match-name")

    linkContainer = gameContainer.find("a")

    matchLink = "https://www.betclic.pt" + linkContainer["href"]

    return matchLink

def getOddsList(matchLink):
    matchSoup = loadSoup(matchLink)

    resFinal = matchSoup.find(id="market_marketTypeCode_Ftb_Mr3")
    oddsMain = getOdds(resFinal)

    resInt = matchSoup.find(id="market_marketTypeCode_Ftb_Htr")
    oddsInt = getOdds(resInt)

    goalsOU = matchSoup.find(id="market_marketTypeCode_Ftb_10")
    if goalsOU == None:
        oddsGoals = [False, False, False, False, False, False]
    else:
        oddsGoals = getOdds(goalsOU)
        oddsGoals = oddsGoals[0:6]

    bttsYN = matchSoup.find(id="market_marketTypeCode_Ftb_Bts")
    if bttsYN == None:
        oddsBTTS = [False, False]
    else:
```

```
        oddsBTTS = getOdds(bttsYN)

    doubleChance = matchSoup.find(id="market_marketTypeCode_68")
    if doubleChance == None:
        oddsDC = [False, False, False]
    else:
        oddsDC = getOdds(doubleChance)
        oddsDC = [oddsDC[0], oddsDC[3], oddsDC[6]]

    drawNoBet = matchSoup.find(id="market_marketTypeCode_Ftb_5")
    if drawNoBet == None:
        oddsDNB = [False, False]
    else:
        oddsDNB = getOdds(drawNoBet)

    oddsList = oddsMain + oddsInt + oddsGoals + oddsBTTS + oddsDC + oddsDNB

    return oddsList

def getOdds(container):
    odds = container.find_all(class_="odd-button")

    returnList = []

    for odd in odds:
        returnList.append(odd.get_text())

    return returnList

def processStats(team1, team2, date):
    match, alt = getMatch(team1, team2, date)

    if match == False:
        return "Stats not available for this match. " +
            teamsNextMatch(nextMatchTeamName(team1),
            nextMatchTeamName(team2)) + "\n  \nWhat else can I help you
            with?"

    if alt == True:
        team1, team2 = switchTeam(team1, team2)

    tip = match.find(class_="tip")
    tip = tip.find(class_="value").get_text()

    allStats = match.find_all(class_="coefbox")

    stat1 = allStats[11].get_text()
    statX = allStats[12].get_text()
    stat2 = allStats[13].get_text()

    stat1HT = allStats[14].get_text()
    statXHT = allStats[15].get_text()
    stat2HT = allStats[16].get_text()

    stat1_5 = allStats[17].get_text()
    stat2_5 = allStats[18].get_text()
    stat3_5 = allStats[19].get_text()

    statBTS = allStats[20].get_text()

    speech = "The fulltime probabilities are: " + stat1 + "% for " + team1 +
        ", " + statX + "% for the draw and " + stat2 + "% for " + team2 +
        ".\n  \n"
    speech += "The halftime probabilities are: " + stat1HT + "% for " +
        team1 + ", " + statXHT + "% for the draw and " + stat2HT + "% for "
        + team2 + ".\n  \n"
    speech += "The probability of the match having more than 1.5, 2.5 and
        3.5 goals is " + stat1_5 + "%, " + stat2_5 + "% and " + stat3_5 + "%
        respectively.\n  \n"
    speech += "The probability of both teams scoring is " + statBTS + "%.\n
        \n"
    speech += "The suggested Tip is to bet on " + tip + "."

    return speech

def getMatch(team1, team2, date):
    dateString = date[0:10]
```

```
    matchLink = "http://www.statarea.com/predictions/date/" + dateString + "/competition"

    soup = loadSoup(matchLink)

    matchs = soup.find_all(class_="match")

    team1 = statsTeamName(team1)
    team2 = statsTeamName(team2)

    match = False
    alt = False

    for m in matchs:
        teams = m.find(class_="teams")
        teams = teams.find_all(class_="name")

        if team1 == teams[0].get_text() and team2 == teams[1].get_text():
            match = m

        if team2 == teams[0].get_text() and team1 == teams[1].get_text():
            match = m
            alt = True

    return match, alt

def statsTeamName(team):
    if team == "Manchester United":
        team = "Manchester Utd"
    else:
        None
    return team

def processTips(team1, team2, date):
    soup = loadSoupTips('https://www.bettingexpert.com/tips/football/england/premier-league')

    container = soup.find(class_="column medium-12")

    matchsContainer = container.find(re.compile("table"))

    matchs = matchsContainer.find_all(re.compile("tr"))

    team1 = tipsTeamName(team1)
    team2 = tipsTeamName(team2)

    matchLink = False

    match = team1 + "-" + team2

    altMatch = team2 + "-" + team1

    for m in matchs:
        details = m.find_all(re.compile("td"))
        matchName = details[1].get_text()

        if match == matchName:
            matchLink = details[2].find("a")["href"]

        if altMatch == matchName:
            matchLink = details[2].find("a")["href"]
            team1, team2 = switchTeam(team1, team2)

    if matchLink == False:
        return "Tips not available for this match. " + teamsNextMatch(nextMatchTeamName(team1), nextMatchTeamName(team2))

    matchSoup = loadSoupTips(matchLink)

    container = matchSoup.find(class_="accordion-content is-active")
    tipsContainer = container.find(re.compile("table"))

    if tipsContainer == None:
        return "No tips available for this match."

    tbody = tipsContainer.find(re.compile("tbody"))

    numbers = tbody.find_all(re.compile("td"))

    home = numbers[2].get_text()
```

```
    draw = numbers[3].get_text()
    away = numbers[4].get_text()

    speech = "The tips for this match are divided the following way: " +
        home + " " + team1 + " bets, " + draw + " Draw bets and " + away + "
        " + team2 + " bets."

    return speech

def processSpecificTip(team1, team2, date, tipster):
    url = 'https://www.bettingexpert.com/user/profile/' + tipster

    print(url)

    soup = loadSoupTips(url)

    container = soup.find(class_="feed-tips")

    matchs = container.find_all(class_="feed-item vevent tip-list-row")

    team1 = tipsTeamName(team1)
    team2 = tipsTeamName(team2)

    for m in matchs:
        matchName = m["data-sort-match"]
        if (team1 in matchName and team2 in matchName):
            altMatch = team2 + "-" + team1

            if altMatch == matchName:
                team1, team2 = switchTeam(team1, team2)

            tip = m.find(class_="medium-5 small-5 column padding-reset")
            tipType = tip.find(class_ = "tip-selection")
            tipName = tipType["title"] + ""

            contents = tipType.contents

            speech = "The tip provided by " + tipster + " is:\n  \n"

            if tipName == "Over / under":
                if "down-dir" in contents[1]:
                    arrow = "Under"
                else:
                    arrow = "Over"

                goals = contents[2][1:5]
                time = getBetTime(contents[3])

                speech = speech + arrow + " " + goals + " goals at " + time
                    + "."

            elif tipName == "Draw no bet":
                team = convertTeam(contents[0][1:2], team1, team2)
                time = getBetTime(contents[3])

                speech = speech + team + " / Draw no bet at " + time + "."

            elif tipName == "Double chance":
                firstTeam = convertTeam(contents[0][1:2], team1, team2)
                secondTeam = convertTeam(contents[2][1:2], team1, team2)
                time = getBetTime(contents[3])

                speech = speech + "Double chance - " + firstTeam + " or " +
                    secondTeam + " at " + time + "."

            elif tipName == "1x2":
                team = convertTeam(contents[0][1:2], team1, team2)
                time = getBetTime(contents[1])

                speech = speech + "Betting " + team + " at " + time + "."

            elif tipName == "Both to score":
                btts = contents[0][16:-1].title()
                time = getBetTime(contents[1])

                speech = speech + "Both to score - " + btts + " at " + time
                    + "."

            elif tipName == "Asian handicap":
```

```
                team = convertTeam(contents[0][1:2], team1, team2)
                handicap = contents[0][8:13]
                time = getBetTime(contents[1])

                speech = speech + "Asian handicap " + team + " " + handicap + " at " + time + "."

            elif tipName == "European handicap":
                team = convertTeam(contents[0][1:2], team1, team2)
                handicap = contents[0][10:-1]
                time = getBetTime(contents[1])

                speech = speech + "European handicap " + team + " " + handicap + " at " + time + "."

            elif tipName == "Ht / ft":
                bet = contents[0][1:-1]
                firstTeam = convertTeam(contents[0][10:11], team1, team2)
                secondTeam = convertTeam(contents[0][14:15], team1, team2)

                speech = speech + "Halftime / Fulltime - " + firstTeam + " / " + secondTeam + "."

            elif tipName == "Half with most goals":
                team = convertTeam(contents[0][23:-1], team1, team2)

                speech = speech + "Half with most goals - " + team + "."

            elif tipName == "Correct score":
                bet = contents[0][1:-1]
                time = getBetTime(contents[1])

                speech = speech + "Correct score " + bet + " at " + time + "."

            else:
                print("else")
                speech = "Tip not available."

            return speech

    return "Tip not available."

def tipsTeamName(team):
    if team == "West Ham":
        team = "West Ham United"
    elif team == "Brighton":
        team = "Brighton & Hove Albion"
    elif team == "Wolves":
        team = "Wolverhampton Wanderers"
    elif team == "Bournemouth":
        team = "AFC Bournemouth"
    elif team == "Leicester":
        team = "Leicester City"
    elif team == "Huddersfield":
        team = "Huddersfield Town"
    elif team == "Cardiff":
        team = "Cardiff City"
    elif team == "Tottenham":
        team = "Tottenham Hotspur"
    elif team == "Newcastle":
        team = "Newcastle United"
    else:
        None
        #team = False
    return team

def nextMatchTeamName(team):
    if team == "West Ham United":
        team = "West Ham"
    elif team == "Brighton & Hove Albion":
        team = "Brighton"
    elif team == "Wolverhampton Wanderers":
        team = "Wolves"
    elif team == "AFC Bournemouth":
        team = "Bournemouth"
    elif team == "Leicester City":
        team = "Leicester"
```

```
    elif team == "Huddersfield Town":
        team = "Huddersfield"
    elif team == "Cardiff City":
        team = "Cardiff"
    elif team == "Tottenham Hotspur":
        team = "Tottenham"
    elif team == "Newcastle United":
        team = "Newcastle"
    elif team == "Manchester Utd":
        team = "Manchester United"
    else:
        None
        #team = False
    return team

def getBetTime(timeContainer):
    if "scope-1" in timeContainer["class"]:
        time = "fulltime"
    else:
        time = "halftime"

    return time

def convertTeam(team, team1, team2):
    if team == "1":
        return team1
    elif (team == "X" or team == "Draw"):
        return "Draw"
    elif team == "2":
        return team2
    else:
        return "Failed at convertTeam"

def switchTeam(team1, team2):

    return team2, team1

def processDate(team1, team2):
    if (team1 not in TEAMS) or (team2 not in TEAMS):
        return "No date available for this match." + teamsNextMatch(team1, team2)

    teamLink = "http://www.espn.in/football/team/fixtures/_/id/" + TEAMS[team1] + "/league/ENG.1"

    soup = loadSoup(teamLink)

    container = soup.find(class_="Wrapper Card__Content")

    monthsContainer = container.find_all(class_="Table2__responsiveTable Table2__table-outer-wrap Table__fixtures")

    date = ""

    team1 = tipsTeamName(team1)
    team2 = tipsTeamName(team2)

    for month in monthsContainer:
        matchesContainer = month.find(class_="Table2__tbody")

        matches = matchesContainer.find_all(class_="Table2__tr Table2__tr--sm Table2__even")

        for m in matches:
            infoList = m.find_all("td")

            if (infoList[1].get_text() == team1) and (infoList[3].get_text() == team2):
                date = infoList[0].get_text()
                hour = infoList[4].get_text()
                break

        if date != "":
            break

    if date == "":
        return "No date available for this match." + teamsNextMatch(team1, team2)
```

```
    speech = "The match between " + team1 + " and " + team2 + " is being played " + date + " at " + hour + "."

    return speech

def processHelp(subject):
    if subject == "Odds":
        speech = "To get odds about a game simply ask for the odds while specifying the match and the date it will occur. For example, \"Tell me the odds for the Chelsea Fulham tomorrow.\""
    elif subject == "Statistics":
        speech = "To get statistics about a game simply ask for the statistics while specifying the match and the date it will occur. For example, \"Tell me the statistics for the Tottenham Wolves tomorrow.\""
    elif subject == "Tips" or subject == "UserTip":
        speech = "There are two types of tip information available: tip summaries and tips from a specific tipster.\n  \nTo get the tips summary for a game ask for tips while specifying the match and the date it will occur. For example, \"Tell me the tips for the West Ham Everton tomorrow.\". \n  \nTo get the tip from a specific tipster ask for the tip while specifying the tipsters, the match and the date it will occur. For example, \"Tell me the tip for the Liverpool Man United tomorrow by tipster name.\""
    elif subject == "Date":
        speech = "To get the date a match will occur simply ask for the date while specifying the match. For example, \"Give me the date for the Liverpool Man United.\""
    elif subject == "Matches":
        speech = "To get a team's next five matches simply ask for that team next matches. For example, \"Tell me Cardiff's next matches.\"."
    elif subject == "":
        speech = "This chatbot provides sports information about odds, statistical predictions, game time and tips for a specified Premier league football game.\n  \nAs for the commands, simply ask the bot for the information you need while specifying the match and the date. To get odds, for example, you can say \"Give me the odds for the Liverpool Man United match tomorrow\".\n  \n For specific help on each subject type \"Help *subject*\", for example, \"Help Statistics\"."
    else:
        speech = "Help Error"

    return speech

def processNextMatches(team):
    if team not in TEAMS:
        return "No next matches available for this team.\n  \nWhat else can I help you with?"

    teamLink = "http://www.espn.in/football/team/fixtures/_/id/" + TEAMS[team] + "/league/ENG.1"

    soup = loadSoup(teamLink)

    container = soup.find(class_="Wrapper Card__Content")

    monthsContainer = container.find_all(class_="Table2__responsiveTable Table2__table-outer-wrap Table__fixtures")

    matchesList = []
    i = 0

    while(len(matchesList)<5 and i < len(monthsContainer)):
        matchesContainer = monthsContainer[i].find(class_="Table2__tbody")

        i += 1

        matches = matchesContainer.find_all(class_="Table2__tr Table2__tr--sm Table2__even")

        for m in matches:
            infoList = m.find_all("td")

            matchesList.append([infoList[0].get_text(), infoList[1].get_text(), infoList[3].get_text(),
```

```
                infoList[4].get_text()])

            if len(matchesList) >= 5:
                break

    speech = team + "'s next matches are:\n  \n"

    for m in matchesList:
        speech += buildMatchString(m) + "\n  \n"

    return speech

def buildMatchString(infoList):

    return infoList[1] + " - " + infoList[2] + " on " + infoList[0] + " at " + infoList[3]

def teamNextMatch(team):
    teamLink = "http://www.espn.in/football/team/fixtures/_/id/" + TEAMS[team] + "/league/ENG.1"

    soup = loadSoup(teamLink)

    container = soup.find(class_="Wrapper Card__Content")

    monthContainer = container.find(class_="Table2__responsiveTable Table2__table-outer-wrap Table__fixtures")

    matchesContainer = monthContainer.find(class_="Table2__tbody")

    match = matchesContainer.find(class_="Table2__tr Table2__tr--sm Table2__even")

    infoList = match.find_all("td")

    return buildMatchString([infoList[0].get_text(), infoList[1].get_text(), infoList[3].get_text(), infoList[4].get_text()])

def teamsNextMatch(team1, team2):

    return "Maybe you meant the next match of one of the teams:\n  \n" + teamNextMatch(team1) + "\n  \n" + teamNextMatch(team2)

def processMatchGeneral(req):
    params = getParamsContainer(req["queryResult"]["outputContexts"])
    team1, team2, date = getParams(params)

    req["queryResult"]["parameters"]["team1"] = team1
    req["queryResult"]["parameters"]["team2"] = team2
    req["queryResult"]["parameters"]["date"] = date

    subject = params["Subject"]

    if subject == "UserTip":
        if params["Tipster"] == "":
            speech = "Who's the tipster?"
            contextName = getContext(req["queryResult"]["outputContexts"][0]["name"]) + "match-general-followup"

            resp = createContextResp(speech, contextName, team1, team2, date)

        else:
            tipster = params["Tipster"]

            speech = processSpecificTip(team1, team2, date, tipster) + "\n\n" + randomFromList(MATCHPHRASES)
            contextName = req["queryResult"]["outputContexts"][0]["name"]

            resp = createContextResp(speech, contextName, team1, team2, date)

    else:
        if subject == "Matches":
            subject = "NextMatches"
            req["queryResult"]["parameters"]["team"] = team1

        req["queryResult"]["intent"]["displayName"] = subject
        contextName = req["queryResult"]["outputContexts"][0]["name"]
```

```
        resp = processRequest(req)
        resp = addContext(resp, contextName, team1, team2, date)

    return resp
```

B

User Questionnaire

The user satisfaction questionnaire the testers answered after completing the tasks. Comprised of eight questions divided into two parts, the first one being the user characterization part, where we collected information about the testers, and the second part where the testers gave their opinion on the system's usability.

Bets Helper

Caracterização dos utilizadores

1. Sexo?
   Mark only one oval.

   - Masculino
   - Feminino

2. Idade?
   Mark only one oval.

   - <14
   - 14-17
   - 18-21
   - 21-27
   - 28-35
   - 35-41
   - >41

3. Quais são as suas habilitações literárias?
   Mark only one oval.

   - Ensino básico
   - Ensino secundário
   - Licenciatura
   - Mestrado
   - Other: _____

Conhecimento desportivo

4. Com que frequência costuma acompanhar desportos?
   Mark only one oval.

	1	2	3	4	5	
Nunca	◯	◯	◯	◯	◯	Todos os dias

5. Com que frequência costuma realizar apostas desportivas?
   Mark only one oval.

	1	2	3	4	5	
Nunca	◯	◯	◯	◯	◯	Todos os dias

Avaliação do sistema

6. Quão fácil foi obter a informação pretendida durante as tarefas?
Mark only one oval.

	1	2	3	4	5	
Muito difícil	◯	◯	◯	◯	◯	Muito fácil

7. Gostou de utilizar o sistema?
Mark only one oval.

	1	2	3	4	5	
Nada	◯	◯	◯	◯	◯	Muito

8. Comparando o Bets Helper com as alternativas existentes, qual foi o seu preferido?
Mark only one oval.

- ◯ Bets Helper
- ◯ Alternativas existentes