Chatbot for Supporting Financial Service Channels

João Francisco Rodrigues

Thesis to obtain the Master of Science Degree in

Telecommunications and Informatics Engineering

Supervisors: Prof. José Manuel da Costa Alves Marques
Engº Tomás Seabra Gomes Braz Teixeira

Examination Committee
Chairperson: Prof. Ricardo Jorge Fernandes Chaves
Supervisor: Prof. José Manuel da Costa Alves Marques
Member of the Committee: Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur

May 2019
To all my Family and Friends, that kept me sane
Acknowledgments

I would like to thank Professor José Alves Marques for giving me the chance to work on this Project, for the opportunity and patience. My advisors, Rui Rasteiro for all the technical help and always being available to help me, and Tomás Teixeira for his support and inputs provided.

I also have a special thanks to my parents for loving me and giving me everything I have, to my close friends for all their strength, support and for their companionship in the last couple years, I will always be there for all of you. And finally my girlfriend, for picking me up when I am down, and for pushing me forward when I do not have the strength, thank you and I love you.
Resumo

Com os avanços da Inteligência Artificial, as máquinas passaram a representar diferentes características humanas, como as entidades conversacionais de inteligência artificial, também chamadas de chatbots.

Os chatbots são programas de computador que podem realizar uma conversa com o utilizador, permitindo que o utilizador simplesmente faça perguntas da mesma maneira que abordaria outro ser humano. Os chatbots estão atualmente a ganhar muita popularidade, especialmente no setor de negócios, pois têm o potencial de automatizar o atendimento ao cliente e reduzir os esforços humanos. Eles cresceram e melhoraram para capitalizar em plataformas de mensagens. A tecnologia principal do chatbot é o processamento de linguagem natural (NLP). Avanços recentes em machine learning melhoraram muito a precisão e a eficácia do processamento de linguagem natural, tornando os chatbots uma opção viável para muitas organizações.

Em 2018, uma nova legislação fez com que os bancos fornecessem um conjunto de APIs seguras para as informações bancárias de seus clientes, tornando organizações bancárias uma das organizações que poderiam beneficiar de chatbots. Neste trabalho, são apresentados os benefícios da inclusão de chatbots, a fim de mostrar o desenvolvimento de um chatbot para aplicações bancárias, que deve ser capaz de realizar operações bancárias solicitadas pelo usuário. O usuário deve ser capaz de se comunicar com o bot, fornecendo ao usuário uma maneira natural, inteligente e simples de executar essas operações, operações que o utilizador normalmente faria em um aplicativo de homebanking. Este projeto foi proposto pela Link Consulting [1] para integrá-lo a outras soluções da empresa.

Palavras-chave: Chatbot, Processamento de Linguagem Natural, BankOnBox, Microsoft, Reconhecimento de Voz
Abstract

With the advances in Artificial Intelligence, machines have started to impersonate different human traits, like the artificial intelligence conversational entities, also called chatbots. Chatbots are computer programs that can carry out a conversation with the user, allowing him to simply ask questions in the same manner that they would address another human. Chatbots are currently gaining a lot of popularity, especially in the business sector as they have the potential to automate customer service and reduce human efforts. They have grown and improved to capitalize on messenger platforms.

The technology at the core of the rise of the chatbot is natural language processing (NLP). Recent advances in machine learning have greatly improved the accuracy and effectiveness of natural language processing, making chatbots a viable option for many organizations.

In 2018, new legislation made banks provide a set of secure APIs access to their customers’ bank account information, making banking organizations be the ones that could benefit from chatbots. This will be presenting the benefits of including chatbots, along with the development of a chatbot for banking applications, which will be able to perform banking operations requested by the user. The user will be able to communicate with the bot, providing the user with a natural, intelligent and simple way to perform these operations, which you would usually do in a home banking application. This project was proposed by Link Consulting [1] in order to integrate it with other solutions from the company.

Keywords: Chatbot, Natural Language Processing, BankOnBox, Microsoft, SpeechRecognition
# Contents

Acknowledgments ........................................... v
Resumo ....................................................... vii
Abstract .................................................... ix
Listings ....................................................... xv
List of Figures ............................................... xvii
List of Tables ............................................... xix

1 Introduction ............................................. 1
   1.1 Motivation ........................................... 1
   1.2 Objective ........................................... 3
   1.3 BankOnBox .......................................... 4

2 Background .............................................. 5
   2.1 Natural Language Processing .......................... 6
   2.2 Speech Recognition ................................... 7
   2.3 State Of the Art, Banking Chatbots ................. 8
      2.3.1 Capital One ...................................... 8
      2.3.2 Bank of America ................................ 9
   2.4 Bot Framework ....................................... 9
      2.4.1 IBM Watson ...................................... 9
      2.4.2 DialogFlow ..................................... 10
      2.4.3 Microsoft Bot Framework ....................... 10
   2.5 BankOnBox API Access ............................... 11
      2.5.1 Authorization and Authentication ............... 11
         2.5.1.1 Oauth2.0 .................................. 11
         2.5.1.2 OpenID Connect .......................... 12

3 Solution ............................................... 13
   3.1 BOB Chatbot Dialogs ................................. 13
      3.1.1 User Authentication ............................ 13
      3.1.2 Balance ........................................ 14
      3.1.3 Movements ..................................... 15
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Results</td>
<td>65</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>69</td>
</tr>
<tr>
<td>6.1 Conclusions</td>
<td>69</td>
</tr>
<tr>
<td>6.2 Future Work</td>
<td>70</td>
</tr>
<tr>
<td>Bibliography</td>
<td>71</td>
</tr>
<tr>
<td>A Listings Appendix</td>
<td>77</td>
</tr>
<tr>
<td>B Chatbot Banking Request Examples</td>
<td>81</td>
</tr>
</tbody>
</table>
Listings

3.1 User's Activity Message Example ............................................. 22
3.2 HTTP POST request example ..................................................... 23
4.1 OpenID client Options and Object creation ................................. 36
4.2 OpenID client function, PrepareLoginAsync ............................... 37
4.3 Chatbot Conversation Reference ................................................. 38
4.4 Defining OpenID handler route .config ....................................... 38
4.5 OpenID Client, ProcessResponseAsync function .......................... 39
4.6 Log in endpoint URL .............................................................. 40
4.7 ASP.NET Web API Routing, for facebook login ............................. 41
4.8 Log in endpoint URL .............................................................. 41
4.9 Log in endpoint URL .............................................................. 42
4.10 Bot Storage configuration code ................................................ 43
4.11 BotFramework Database Client Functions ................................ 45
4.12 SQL Query, Searching Document by "facebookid" ......................... 45
4.13 Snippet from BasicLuisDialog.cs, initially created by the SDK, which a few changes ...... 51
4.14 Snippet of LUIS’ prediction response ...................................... 51
4.15 OpenID Connect, Refresh Token ............................................... 52
A.1 MessageController.cs code snippet, automatically created by the SDK ...... 77
A.2 Snippet from BasicLuisDialog.cs, initially created by the SDK, which a few changes ...... 78
A.3 LUIS response ................................................................. 78
A.4 Chatbot Webpage, HTML File. .................................................. 79
A.5 BOB API, HTTP POST example, Payment Request ........................ 80
A.6 BOB API, HTTP GET example, Account Balance Request ................ 80
List of Figures

3.1 Chatbot Payment Dialog Usage Example ............................................................... 14
3.2 Chatbot Balance Dialog Usage Example ................................................................. 15
3.3 Chatbot Movements Dialog Usage Example ............................................................ 16
3.4 Chatbot Payment Dialog Usage Example 1 .............................................................. 17
3.5 Chatbot Payment Dialog Usage Example 2 .............................................................. 18
3.6 Chatbot Payment Dialog Usage Example 3 .............................................................. 19
3.7 BOB Chatbot Architecture ....................................................................................... 20

4.1 Saldo Intent, Utterances and Score Examples ......................................................... 31
4.2 Saldo Entities, Labeled vs Unlabeled ..................................................................... 31
4.3 OpenID Connect Connection Flow ....................................................................... 39
4.4 OpenID Connect Connection Flow ....................................................................... 42
4.5 "Account Collection Document Example" ............................................................. 44
4.6 "Bot Collection Document Example" ....................................................................... 46
4.7 "State Collection Document Example" .................................................................... 46
4.8 Chabot Interface Usage Example ............................................................................ 47
4.9 Dialog Flow, Login ................................................................................................. 54
4.10 Dialog Flow, Saldo ................................................................................................. 56
4.11 Dialog Flow, Consultas ......................................................................................... 58
4.12 Dialog Flow, Pagamento part1 .............................................................................. 61
4.13 Dialog Flow, Pagamento part2 .............................................................................. 62
4.14 Dialog Flow, Pagamento part3 .............................................................................. 63
List of Tables

5.1 LUIS Intents Training Results ............................................. 65
5.2 LUIS Entities Training Results ............................................. 65
Chapter 1

Introduction

1.1 Motivation

Chatbots represent a potential shift in how people interact with data and services online. The most intuitive definition for a Chatbot is, a software that can have a conversation with a human. They can perform a wide variety of functions based on user commands, often used online and in messaging apps, as intelligent virtual assistants.

Text communication nowadays takes a huge part in people’s lives. Almost all age groups use text messaging for personal, family and social communication as well as for business purposes, many companies are even using text messaging for communication between employees.

According to comScore’s 2017 Cross-Platform Future in Focus report [2], the average American adult (18+) spends 2 hours, 51 minutes on their smartphone every day, how much of that time is spent texting? eMarketer released a study in 2016 which showed that 22% of the time spent on the phone is spent texting [3].

One of the most known and used messaging app is Facebook’s Messenger [4], and they saw great potential for chatbots, ever since they launched the Messenger Platform, which is nothing more than a toolbox for building bots [5], Facebook already has over 100,000 bots developed by its users, allowing companies to provide customer assistance and relationships through conversation, users are using a communication channel they are familiar with, texting, to have a conversation with a bot, they are already used to the technology, Facebook VP of messaging products David Marcus said, “People prefer to use Messenger to interact with companies”, during his keynote at the F8 2016 conference [6].

More and more companies are complementing their businesses with chatbots, users necessities, customs, and expectations are always changing, customers always become more demanding, and with the accommodation of message applications, Facebook’s Survey shows “53% of people are more likely to shop with a business they can message” [7]. Those who comply with the demands are already seeing an increase in profits, take Sephora as an example, which has seen an 11% increase in booking rates through the Sephora Reservation Assistant and an average spend of over 50$ from clients who have booked an in-store service via its Messenger assistant [8].
Regarding what nowadays people expect from businesses in terms of availability, "51% of people say a business needs to be available 24/7" [9], no one enjoys waiting endlessly on the phone, being put on hold, or having to push buttons to navigate from department to department, just to find the answer to a question that a bot can resolve in 30 seconds and at any time, without having to worry about working hours. If the world moves one step further, it’s possible to change call centers to be fully automated [10]. Of course, this brings a lot of different issues, but on the brighter side, these systems need no human interference, which is unpredictable and not always accurate, humans are restricted to the number of things we can do at the same time. A study suggests that humans can only concentrate on 3–4 things at the same time, “For example, when we present phone numbers, we present them in groups of three and four, which helps us to remember the list” [11]. If it goes beyond that you are bound to meet errors, chatbots, on the other hand, can simultaneously have conversations with thousands of people. No matter the time of the day or how many people are contacting you, every single one of them will be answered instantly, in fact, reports show that the vast majority of chatbot users “reported productivity to be the main reason for using chatbots. These participants highlighted the ease, speed, and convenience of using chatbots.” [12].

There are many anticipated benefits for Chatbots:
- Helps you Save Money: Chatbots are a one-time investment, which helps businesses reduce staff required and possibly also bringing new customers. Unlike human beings, chatbots work 24/7. Chatbots can be fast enough to provide this kind of predefined service with less error compared to a human [13].
- Provides 100% satisfaction to customers: Humans react to others based on their mood and emotions. If an agent is having a good attitude he will most probably talk to customers in a good and polite way but if he is having a bad day he may be more rough to this customer and the customer may not be satisfied. Whereas chatbots are bound by some rules and obey them as they’re programmed to. They always treat a customer in the most polite and perfect way no matter how rough or rude the customer is [14].
- Personalized Experience: When you visit a website searching for a piece of information, you know it’s probably somewhere in there and you will end up having to scan around different web pages and menus to find what you are looking for [15]. Instead with a chatbot, the interaction is resolved to the questions you make. Also, Chatbots can change to accommodate the users’ preferences (an easy example of this is that the Chatbot could communicate in any language preferred by the user.)

One of the business sectors that could benefit from chatbots are financial systems. There was a time when Banks would keep all the data and information regarding their customers, which could only be accessed via the bank on-line portal. However, this has recently changed with new legislative rules. On the 16th of November of 2015, the European Union passed the Revised Payment Services Directive (PSD2 – EU Directive 2015/2366) [16], giving member states two years to incorporate the directive into their national laws and regulations. Finally, on the 13th of January of 2018, PSD2 entry into force into local legislation, companies now had a minimum of 18 months, before PSD2 rules were mandatory.

In short, PSD2 obligates banks to provide third-party providers access to their customers accounts through open APIs, the API provides a secure avenue that allows bank account holders to access their banking data and services by granting access to third-party applications appropriately authenticated by
the national regulators, following the consent of both the bank and customer.

That is really interesting because it means that now new providers, not necessarily banks, can consolidate your accounts information in one place and acquire insightful data for you. With the banks APIs, a single app can have access to banking information related to checking accounts (credit, debit, etc.), with contents like balance, movements, transfers, everything except information that is deemed sensitive and can directly identify the end user. This will lead to the appearance of multiple applications to ease access to our account information and make online payments without credit cards, i.e. direct access to our bank account via the Internet.

Many of these apps are expected to start emerging, this offers lucrative cross-selling opportunities for these new providers, you may soon be using Facebook or Skype to check your account balance or making payments.

1.2 Objective

The main goal of this thesis is the development of a chatbot that would complement the current way to access the BankOnBox Home Banking solutions, presented in the next section 1.3.

Instead of browsing through a website, you will have a conversation with the Chatbot. The user should be able to communicate with the chatbot, so he can ask/order it to do any of, but not limited to, the common banking operations you would do in applications that used the BankOnBox solutions, which provides ways to do banking operations, but instead of having to go to your financial institute application or having to access different applications for different banks, you can do these operations all in one application, the chatbot, that will provide access to all the financial institutions that use the BankOnBox solutions.

The bot is expected to support and only be used in Portuguese, this is a key feature, because it defines the environment in which this application should be used, because not every system available can recognize this language, so it will have to be taken into consideration when choosing the technologies that will be used in this project, they have to support the Portuguese language.

To offer the customers a different and more intuitive way to interact with this technology, the user will be able not only to communicate via text but also by voice.

To fulfill these objectives, part of the research falls upon learning about Chatbots, what technologies are behind implementing a chatbot, searching for already existing solutions and gathering well-formed opinions and ideas to better present a solution to the problem at hand.
1.3 BankOnBox

BankOnBox [17] is a solution for Financial Institutions, developed by Link Consulting (Link) [1], which was designed to meet the needs of the Banks and of their customers. In the context of this project, the focus will be more on the functions of Internet Banking, also known as Home Banking, which allows the clients to perform the traditional banking functions (consultation of balance, transfers, payments, etc.), through the financial institutions’ website. BankOnBox basically provides solutions for Financial Institutions to implement Home Banking in their system. BankOnBox Self-Banking solutions provided financial institutions with different channels for their users’ remote banking needs, like a Self-Banking website, to be used on computers, or WebMobile, a web application designed for smartphones, but these channels must belong to the financial institutions because only they have access to the banking operations. With the upcoming changes in EU legislation [16], as described in the Introduction 1, the banks are being forced into opening themselves up and allowing their customers to provide access to third-party applications to their bank accounts, within the scope of consent that was provided by the end user.

That is basically the concept of Open Banking [18], banks must look at opening their clients’ banking information to third parties through Application Programming Interfaces (APIs). APIs are software-to-software interfaces that enable partners and consumers to perform business functions and to securely exchange data which is otherwise confined within a proprietary system, in this project’s context, the chatbot should be able to freely access the banks’ API, to access banking information and also perform banking transactions.

Link is aware of this and is already developing a solution to implement Open Banking for financial institutions that use BOB’s technologies. The BOB API is a thesis project that was developed by João Silva Santos, which consists of implementing a set of APIs which are PSD2 compliant, previously mentioned in section 1.1, and provide the minimum necessary functionalities. These functionalities will, in the end, let every user be able to access his banking information from applications other than the one provided by his financial institution. The API will provide access to the same main features that BOB Home Baking has, which will have to match the bot features:

- Access to a user’s bank account(s) balance;
- Access to a user’s bank account to initiate payments;
- Access to a user’s bank account to movements transfers information;
Chapter 2

Background

A Chatbot is a computer program, powered by a set of rules and some artificial intelligence, that you can interact with via a chat interface, chatbots are designed to imitate human conversations and carry out actions, based on voice commands and/or text inputs [19].

Chatbots have been around for a long time, the idea of chatbots is as old as the computer itself. It was introduced for the first time by a creator of theoretical computer science Alan Turing in his seminal paper "Computing Machinery and Intelligence" in 1950 [20]. In the paper, Alan introduced the concept of the Turing Test, which would test if a computer can act indistinguishably from the way a thinker acts. In this test, a judge should decide if it is talking to a human or a machine at that moment. The job of the judge would be to interrogate the human and machine with a series of questions and based on their reactions, tone, and how they reply to the questions being asked, make a decision. The very first known chatbot was Eliza, which was developed in 1966 [21]. Its goal was to behave as a Rogerian psychologist. Its conversational ability was not very good, but it was enough to confuse people at a time when they were not used to interact with computers.

Over the past decade quite a bit of advancements have been made in the area of chatbots, major Internet companies such as Google, Facebook, and Microsoft now see chatbots as the next popular technology; Microsoft CEO Satya Nadella stated [22], “Bots are like new applications, and digital assistants are meta apps or like the new browsers. And intelligence is infused into all of your interactions. That's the rich platform that we have." Chatbots are highly used to provide a variety of different services, as it would be expected, one of the biggest examples right now is Facebook Messenger. Imagine you are looking to buy a T-Shirt from your favorite clothing store, if this store has (which most likely does) a Facebook page, you could simply message it, asking for exactly what you want, and the chatbot would provide you with a list of options for you to choose from.

After understanding what a chatbot is, I had to get an idea of what a bot is made of i.e. the components used in making a chatbot. There are basically two kinds of chatbots, first you have the bots that limit the user interaction, like the Q&A bots, Question and Answer bots, these bots do not allow the user to ask questions, instead they are the ones that make the questions to the user and, at the same time, provide the user with a list of possible answers for him to pick from, for each option it gives it has
a corresponding answer, everything is predefined, the problem with this bot is that they do not allow the user to really say anything, they just pick options.

Then there are bots that risk trying to parse anything the user says, those who are equipped with natural language processors, these bots can give the user the feeling that they’re having a conversation with a real person, rather than poking through tedious text options. These type of chatbots are designed to understand the user intent and keep a conversation with different persons, every person talks in a different way, for example: If I am trying to know my current balance I might say: “What's my current balance in my savings account?” and someone else might just say: “How much money have I saved?” These two phrases have the same intent, knowing the balance on the savings account, but since they were said in such a different way, these types of Chatbots would not be able to understand that, they should be used in scenarios where static or predefined answers work, but as you can see this is not the case, the users should be able to be themselves, to talk like they usually do, without having the feeling they were better off talking with a real person. For the context of this project, this is the type of bot that is to be implemented in this project.

2.1 Natural Language Processing

Natural Language Processing (NLP) is an integral part of any smart chatbot, perhaps the most important component of a chatbot. So, I knew from the start that I would require AI and a strong NLP system to make my bot “smart” and something you could talk to.

The arrival of talkative assistants like Siri, Alexa and Cortana are all well and good, but there’s no point if the device or service does not understand what the person is asking for, or talking about. Over the few years of their short lives, they have become better at understanding people thanks to NLP, a technology that learns words and phrases and their meaning to provide a better service.

In a spoken dialog system, natural language processing (NLP) converts the words in an utterance into a machine-readable meaning representation. [23]

Natural language processing (NLP) is a branch of artificial intelligence that helps computers understand, interpret and manipulate human language, it can also examine and use patterns in data to improve a program’s understanding, it is basically the ability of a machine to analyze and understand human speech. Because computers operate on artificial languages, they are unable to understand natural language, this is the problem that NLP solves. With NLP, a computer is able to listen to an utterance spoken by a person and understand its meaning. Of course, there are several complex steps involved in that process. While natural language processing is not a new science, but in this past years, the technology is rapidly advancing thanks to an increased interest in human-to-machine communications, plus an availability of big data, powerful computing, and enhanced algorithms.

To understand how NLP works, we have to take a look at one of its main components, Natural Language Understanding (NLU), the most difficult part of NLP is understanding, or providing meaning to the natural language that the computer received, that is NLU’s job. Natural Machine Understanding deals with machine reading comprehension, to understand the meaning of a text. NLU uses algorithms to
reduce human speech into a structured ontology, although different NLU systems use different techniques, the process is generally similar. First, the computer must understand what each word is, it tries to understand if it's a noun or a verb if it's past or present tense, and so on. This is called Part-of-Speech tagging (POS) [24]. Then they have a vocabulary and a set of grammar rules, statistical machine learning algorithms apply these rules to the natural language and determine the most likely meaning behind what was said. These are the main components to understand NLP, with the right training the system should be able to understand the meaning of what you said.

The main drive behind NLP is to create chat and speech-enabled bots that can interact effectively with the public without supervision. NLP technologies are becoming increasingly relevant in different software areas today including both technologies, NLP is a pursuit of many start-ups and major IT companies, like IBM, Microsoft, and Google, which developed services that allow developers to implement NLP on their applications.

Watson's Conversation Service (WCS) is especially focused on automating interactions between systems and end users. Using WCS users can define NLP aspects such as intents, entities and simulate entire conversations. WCS is typically used in conjunction with other Watson NLP services such as Alchemy Language or Natural Language Classifier.

Microsoft's Language Understanding Intelligence Service (LUIS) is a component of the Microsoft Cognitive Services focused on creating and processing natural language models. LUIS provides a sophisticated toolset that allows developers to train the platform in new conversation models. The platform provides deep integration with Microsoft's Bot Framework and can be used by other bot platforms.

Google Natural Language (NL) API is a recent addition to Google Cloud focused on NLP. The NLP API enables capabilities such as intent-entity detection, sentiment analysis, content classification, and relationship graphs. It also includes sophisticated tooling for training and authoring new NLP models.

## 2.2 Speech Recognition

Speech is one of the most powerful forms of communication between humans, speech recognition software is used to convert spoken language into text by using speech recognition algorithms.

In natural speech there are hardly any pauses between successive words, also that in most spoken languages, the sounds representing successive letters blend into each other in a process termed co-articulation, so the conversion of the analog signal to discrete characters can be a very difficult process. Thus speech segmentation is a necessary subtask of speech recognition. Segmentation of Speech separates the waveform into principal acoustic units, i.e. the process of breaking down a speech signal into smaller units.

An then there is another problem, given that words in the same language are spoken by people with different accents, the speech recognition software must be able to recognize the wide variety of input as being identical to each other in terms of its textual equivalent.

Knowing that, in speech recognition, there are usually three common models that work together [25].

- The acoustic model takes the waveform of speech and chops it up into small fragments and figures
out each sound that the person is speaking.

- The pronunciation model that takes those sounds and strings them together to make words
- The language model that takes the words and strings them together to make sentences.

Speech Recognition can be used by people with disabilities, for in-car systems, in customer service, etc, voice may be the next major banking channel. The main problem with speech-enabled applications is having an interaction that feels natural to customers, where they can speak as they would to another human and not speak as if they’re talking to a robot, but 2017 has been an excellent year for Voice Interfaces, voice experiences are going mainstream due to the effort from corps like Amazon and Google to pushing forward the Smart Speakers market with Alexa, as well as the Human-Like robot Sophia which spent all year traveling all around the world, talking to numerous people. Companies are opening their speech recognition solutions, providing APIs or web services so that developers can bring speech recognition to their own apps and services. Yet again, the best current provider of speech recognition solutions is Google, with Google Cloud Speech Recognition API, considered the current best speech technology [26], but unfortunately the most expensive, next is IBM, with IBM Watson Speech-to-Text, and finally Microsoft, with Microsoft Cognitive Services Bing Speech API.

2.3 State Of the Art, Banking Chatbots

The chatbots have made life simpler for humans. The way they are designed and with the advancements in NLP, chatbots are capable of assisting us with almost everything. We can notice chatbots marking their presence around us. Particularly in the banking industry, it is changing the face of the communication interface by adopting Artificial Intelligence. Introducing chatbots in the banking sector can bring a huge change in customer experience and keep up the pace with changing customer expectations. banks are now stepping forward to use chatbots to simplify the overall banking experience for the customers making them stand ahead in the competition but also your organization.

2.3.1 Capital One

Chatbots have now entered the credit card market. Capital One Financial designed a chatbot called “Eno” - “one” spelled backward [27]. This bot can interact with the bank’s customers through text messages. It can provide information on customers’ accounts and help them make credit card payments from their smartphones. Eno even understands certain emojis like thumbs up and thumbs down, as well as the money bag. And, no need to be formal when texting Eno, you can write “bal” and it will know to check your balance. Currently, no other major issuer has a service to compete with Eno, giving Capital One a leg up on competitors Since Eno can already perform a variety of common credit card actions that you’d typically have to access online or via a mobile app after logging on, as well as add extra security for online purchases, it’s evident there is value for consumers. And the future looks promising for Eno, seeing as the world becomes more technologically dependent, with consumers preferring minor tasks, like paying a bill, to be as quick and hassle-free as possible, and to be done from anywhere at any time.
2.3.2 Bank of America

As a market leader in both mobile banking use and AI implementations in the U.S., Bank of America introduced Erica [28], to send notifications to customers, provide balance information, suggest how to save money, provide credit report updates, pay bills and help customers with simple transactions. Since the introduction, the capabilities of Erica have expanded as an advanced virtual assistant to help clients make smarter decisions.

Erica is available inside the Bank of America mobile banking app. Customers can get help via voice or text regarding their banking problems. This banking chatbot sends personalized recommendations, offers, and advice after analyzing the customer’s data. Erica can also send educational videos on finance.

2.4 Bot Framework

Bot framework is a set of predefined and pre-installed methods and classes created for bot developers. It provides developers a set of tools that help them write the code better and faster. In simple terms bot developers and programmers use development frameworks so they don’t have to create chatbots from scratch [29], saving learning and development time, with security measures implemented for you and documentation to ease functionality implementation. These days, there are so many platforms offering a bot framework for building chatbots and each one has varying degrees of learning curves. You need somewhere in between 1 to 5 hours to understand a platform depending on your technical know-how. Once you have the proper understanding, you can build a basic chatbot in minutes, without having to worry about the hosting environment for the chatbot and you do not need to do any brainstorming for server configuration. All these bot platforms operate on the cloud, over the Internet, being cloud-based will make the application be much lighter, it will not spend energy and local resources, which is really important for mobile apps that run on a resource wise, limited machine, surely, the machines hosting the chatbot have higher CPU than any mobile device has, probably making it even faster than running it locally on the device. I studied the currently most famous and modern bot platforms, which allows developers to create their own bots, to later select the best one for this project. Those platforms are IBM Watson, Dialogflow developed by Google and Microsoft Bot Framework.

2.4.1 IBM Watson

According to the research study by Mindbowser in association with Chatbots Journal [30], IBM Watson is the first choice as a bot-building platform for 61% of businesses. Watson is an AI platform capable of answering questions posed in natural language. Watson’s strongest suit is the Conversation Service that is built on a neural network (one billion Wikipedia words), which is responsible for understanding natural language and machine learning. The computer system was initially developed to answer questions on the quiz show and last year Watson was being prepared to treat breast and lung cancers [31].

Watson provides SDKs (Software Development Kits) for Node.Js, Java, Python, iOS, and Unity. It is
easy to start development and allows us to deploy bots on messaging platforms, mobile devices, and even robots. The data on the bot is also secure as the platform allows users to opt out of data sharing. It also offers speedy integration with a wide array of networks, channels, and environments.

One of the most important components for a chatbot is the natural language processing, and Watson capabilities are being extended to provide new usages, that is probably why such a high percentage of businesses choose IBM has their chatbot platform. Unfortunately, at the time we started this project, IBM Watson only supported English and Japanese, which automatically excluded this as a platform option, since one of the objectives is for the chatbot to work in Portuguese.

2.4.2 DialogFlow

DialogFlow is the Google-owned Chatbot platform, it incorporates Google’s machine learning expertise and products such as Google Cloud Speech-to-Text, allowing us to understand the user and also “hear” him. It was previously known as Api.ai and was renamed after Google bought it in September 2016. Dialogflow uses information like examples, contexts, annotations, etc. to match the user queries to a suitable intent. Dialogflow provides a platform that allows developers to design and implement conversation interfaces that can be embedded in external bot applications. It currently has SDKs for Android, iOS, Cordova, Unity, Xamarin, HTML, JavaScript, Node.js, Epson Moverio, Botkit, .NET, C++, Python, Ruby, PHP, and Java. A total of 14 languages are supported, which includes Portuguese.

The strength of DialogFlow is basically Google’s expertise, they provided the platform with content that they spent years developing, Google’s Speech-to-Text API is one of the bests, that also works with numerous languages. Sadly, Google falls behind in the natural language understanding, an experiment showed that [32], between Microsoft LUIS, Watson Conversation and DialogFlow, the latter had the lowest Recall value, that represents the percentage of relevant instances retrieved out of the total set, meaning it had many false negatives, instances that were considered not relevant, but were in fact relevant, for natural language it means that the system was not able to correctly identify the user request.

2.4.3 Microsoft Bot Framework

41% of the businesses in Mindbowser’s study [30] said they prefer Microsoft Bot Framework [33]. It has its own Bot Builder SDK that includes the .NET SDK and Node.js SDK. The entire system consists of three parts: Bot Connector, Developer Portal, and Bot Directory. The framework provides the Direct Line REST API, which can be used to host a bot in an application or website [34]. It is possible to incorporate Microsoft LUIS for natural language understanding and the Bing APIs for speech recognition. LUIS scored as the best Language Understanding system [32], behind the overall most accurate processor over the options above. Both LUIS and the Speech API support the Portuguese language. Another positive aspect is, it is open source and available to all on GitHub [35], which in my opinion is equal to bigger documentation, which is essential for an easier development of applications, which is the main purpose of these platforms, to make the developer’s job easier.
2.5 BankOnBox API Access

For the bot to be able to perform any banking operations, it has to be able to communicate with the BankOnBox API, BOB API, which defined a way to authenticate and/or authorize users and only letting them access resources that are within their scope of authorization, also making sure that the application trying to access the API is trustworthy, for that the client should authenticate itself in the system first. This authentication was defined in the BOB API.

2.5.1 Authorization and Authentication

Authentication is about obtaining and validating identification credentials, such as username and password, although there are other various ways to be authenticated. The system checks whether you are what you say you are using your credentials. In case these are valid, follows an authorization process in order to comply with PSD2 [16], once the user is authenticated, the authorization process determines which permissions the target application will have.

Authorization is, as just mentioned, the process to determine whether the authenticated user has access to the particular resources. In other words, it is about obtaining consented access to the user’s information via the current service. This consent is usually given by providing an object which contains information of the consent that the user gave along with the user request (e.g. an Access Token).

In the context of this project, these properties will be implemented by using OAuth2.0 and OIDC protocols, to provide authorization and authentication needed for the users.

2.5.1.1 OAuth2.0

OAuth2.0 is an open-standard authorization protocol or framework that describes how unrelated servers and services can safely allow authenticated access to their assets without actually sharing the initial, related, single login credential. Commonly used as a way for Internet users to grant websites or applications access to their accounts but without giving them the passwords. Instead, users are redirected to the web service authentication page after authentication is granted, what the application gets is, what is called as, an access token that allows access to the user’s resources, but expires after a given time. Also if the user gave permission for a refresh token, that allows the application to re-authenticate the user, by receiving a new access token, without requiring the user to re-login.

This, however, does not authenticate the end user, Authentication in the context of a user accessing an application tells an application who the current user is and whether or not they are present. A full authentication protocol will probably also tell you a number of attributes about this user, such as a unique identifier or an email address. However, OAuth says absolutely nothing about the user, nor does it say how the user proved their presence. As far as an OAuth client is concerned, it asked for an access token, got a token, and eventually used that token to access some API.

But there is actually a way to use OAuth 2.0 to perform user authentication, which is why an Identity Layer that uses OAuth2.0 exists, namely OpenID Connect (OIDC) [36].
2.5.1.2 OpenID Connect

OpenID Connect (OIDC) is an open standard published in early 2014 that defines an interoperable way to use OAuth 2.0 to perform user authentication. OIDC is a protocol used to enable clients to authenticate that the user is who he says he is when requesting initial access and permissions to be consented for in regards to a service’s account. While OAuth is only concerned with providing authorization to resources that the user consented to, OIDC also ensures that the end user is authenticated with the server and application, alongside the OAuth access token, the application receives the user’s identity encoded in a secure JSON Web Token (JWT) [37], called an ID token, in which the user’s claims are packaged. Each application should define which specific claims they use, including some way for the server to identify its users, either a value that is unique in the context of the issuer or that is globally unique.

The chatbot should be able to store these tokens and use them accordingly, for user authentication and having authorized access the user’s banking information from BOB API, and comply with any user request, so that the client does not have to authenticate every time that the bot wants to access the BOB API, while the user is using the system. With access to the refresh token, the chatbot will also be able to refresh the access token for the client without needing to ask the user to re-authenticate, while the user is still using the system.

These tokens have a lot of power, the API has to make sure that the system protects the privacy and integrity of this token exchange, but when the user authenticates on the BOB API, the system requests the user to define the permissions it wants to give the application the user is using to connect to the API. The user is informed of the type of information permissions that the application is requesting to access, in the BOB API case, the user is asked to specify what type of permission he wants to give to the application. Although you giving these apps access to your bank account, it will be in read-only mode. These bots can only view what is going on in your account and report back to you, and any cash transaction can only be done if the user consents to it, if the transaction was requested by the user, for example, making a payment, the chatbot should present the transaction, so that user can verify it is correct and ensure that the user wants to make that transaction or not. The other option is, if the transaction was requested by the bot, such as transferring some money to the user savings accounts, the chatbot would send a notification to the user, asking if he allows the bot to do that for him or not.
Chapter 3

Solution

In this section our chatbot solution will be presented, first an introduction to the chatbot, the supported chatbot dialogs that the chatbot should have to understand user commands, the language understanding capabilities that are expected from the chatbot. After that the chatbot architecture will be presented, where the choices for each of the chatbot components, studied in section 2, will be explained.

3.1 BOB Chatbot Dialogs

This chatbot is designed to perform the banking operations available in BOB API, account balance, movements transfers, payments, that were described in section 1.3, and to access BOB API, in section 2.5, the user authentication. The bot must have a dialog prepared for each of these actions, meaning that depending on the user message, the chatbot must respond according to the request. Dialogs define what the chatbot can expect from the user, what it has to do, depending on the user message, and what should be answered.

To be able to perform these operations, the user must first be able to authenticate itself into the system, after that, natural language processing will be used to understand user inputs, to understand user intentions and decide which of the supported dialogs will be triggered.

The following sections will explain each of the supported dialogs, as well as showing off what natural language processing capabilities are expected for this chatbot, what types of requests will the user be able to do and also how the chatbot will handle BOB API responses.

3.1.1 User Authentication

Figure 3.1 represents the Authentication Dialogs and the two possible outcomes of user authentication for this solution

In the beginning, the user has to start the conversation with the bot, independent of what the user says, when the user sends the first message the bot will trigger the opening login card, if the user does not choose to authenticate, but rather send another message, card will keep repeating until the user logs in. The user has two options authenticates into the system, as explained in section 4.6.5, the left section
of Figure 3.1 shows the response results for both cases, in case of BOB login, the user will always be presented with the same message when it authenticates, the name shown depends on the returned user Claims, from Openid Connect in section 4.2.1. The same message will appear with the Facebook Login, if the user had already authenticated himself into the system previously, on the contrary, if it is the first time, the user will be faced with the result from the right section of Figure 3.1, where the user is prompt to link its BOB account to the just authenticated Facebook’s identity, which when accepted the user will be taken to BOB’s Login and re-authenticate himself, which when done successfully will trigger the same message as the first case.

3.1.2 Balance

Figure 3.2, represents the possible responses from the balance operations and the language processing capabilities of this solution, to detect other user intentions.

The first message is the simple balance request, 1, which when LUIS identifies as having to specific requests entities, returns the list of the user’s account, 2. The user can click any of the accounts to pick it, from clicking in each of the different list accounts, you get a message for each corresponding account and its balance, 3, and the option to show all the balances at once, “Mostrar Tudo”, returns a message with all the accounts balance at once, 4.

The next message shows the language processing implemented capabilities where the user can ask the chatbot to show the balance from the bank account it wants, 5, the chatbot was able to correctly identify a request for a specific account balance, the user asks for the “new generations” account, the bot sends the user a notification saying that it is currently performing the balance request and the correct account balance was then shown, 6, as can be seen by the presented account name.

After that, a different account was tried, in this case, the balance for the account “banif”, 7, which like the previous, also returns the notification and the requested account balance, 8.
3.1.3 Movements

With Figure 3.2 it will be shown some of the different movements requests that the user can perform and the corresponding results of the chatbot solution.

The first message is the simple movement’s request, 1, which will warn the user that it is performing the requested action and when it gets a response from BOB API, returns the full transfer movements listed by date, 2.

The next option available for this dialog is to ask for the movements of a specific account, 3, in this case, the “new generations”. The bot then returns the list of movements from the correct account, the name of the account is shown at the beginning of each movement listing, 4, which has different transfers than the previous one, 2.

The system should handle all different accounts, now the “Savings” account, 5, for which the movements are then listed, 6, and this also shows that the bot was able to access all 3 of the users accounts and showing their movements correctly (In Figure 3.2 it can be seen the list of users accounts, in 2).

But these previous operations only showed the list of transfers that were sent from BOB API, but the user should also be able to filter the results of the movement’s request. The user is able to filter their movements by date, any requested date and the bot will filter the list of transfers, according to the request.

The user asks for the movements from April 2012 of the new generations account, 7. For this utterance, LUIS had to identify the account in the utterance in order to make the API call for the requested account, as well as the requested date of the requests in order to filter the transfer movements received from the API operation. In 8, we can see that at the top the bot correctly identifies that it is showing the results for 2012/04, April 2012, then the account that is being shown is indeed the “new generation”
account, and finally the movement’s list was filtered, only showing the results that were either Requested or Concluded in April 2012. You can also compare the results from 4 and 8, to confirm that the filter was correctly performed.

3.1.4 Payments

This section is defined into 3 segments, in which it will be analyzed different figures, each representing the Payment Dialogs of the solution.

First is Figure 3.4, the user makes a payment request, in 1, which sends the user a blank payment card, 2, which has the necessary fields to make a payment, it also warns the user that it is going to need
to fill the IBAN and the money amount to complete its payment.

The user is never obliged to respond so it can decide to write a new sentence. The user should not have to enter the values in the card every time, if the user knows what it wants to transfer, it should be able to just say it.

First the chatbot should be able to understand money amounts, in this example the user asks to make a payment (or transfer), with the desired amount of fifty euros ("cinquenta euros"), 3, the bot then proceeds to warn the user that it only needs the IBAN to complete the request, since it already has the amount, to send the payment card to the user, but this time, filled with the requested amount, 4.

Also, payments are only done through using the IBAN number, as defined in the BOB API, so the chatbot needs to correctly identify IBAN numbers in an utterance. The user can ask for a payment and
also provide the IBAN number in the request, 5, which will create a payment card where the IBAN value is already with the IBAN that the user asked for, 6, and tell the user that it only needs the amount this time.

Finally, the chatbot should be able to understand a complete payment request, the user can also provide both the amount and IBAN number, in a single request, 7, the bot will then fill both fields accordingly and tells the user to finish the payment by providing a description for the payment, 8.

Figure 3.5: Chatbot Payment Dialog Usage Example 2

Figure 3.5 represents how to complete the payment, the user proceeds to fill in the payment description and confirms payment, 1. The bot starts the payment operation, if the payment is successfully started it will warn the user and ask him to introduce the SMS code that it would receive for the associated phone number, to confirm that the user wanted indeed to make a payment, and sends the payment confirmation card, 2, the payment card contains the information of the payment you just confirmed to as well as the account from which the operation will be performed.

When the user enters the code, it can confirm the payment, 3, this will start the payment completion operation, which the chatbot will warn the user if it successful, and, in this case, a new IBAN was used,
the bot sends a card asking if the user wants to add the new IBAN to the Contact List, 4. This option will allow storing user contacts so that it can make payments to a specific user instead of having to write the full IBAN value every time.

If the user decides to give a name to the contact, in this case, “jose”, 5, the contact will be added to the user’s contact list for future use.

Lastly is Figure 3.6, where it will be shown the last features of the payment dialog.

After adding a new contact, you can start using it instead of using the IBAN number, so in the first request, the user asks for a payment of four hundred euros to the previously created contact, “jose”, 1. The bot sends a message that says that to complete the payment to “jose” it only needs the description and follows by presenting the pre-filled payment card, with the corresponding IBAN that was saved as the contact “jose” (which can be compared with Figure 3.5 saved IBAN, 5).

The user can check the card at any time to see the account it is using to make the payment, represented by “Utilizando a conta:”, and also the corresponding balance for that account, represented by
“Saldo de conta: “. If the account is not the one that you the user wants to use, or it does not have enough balance to make the payment, the user can use a different account than the select one to make the new payment, the user can click in “Mudar de Conta” and select the account it wants to use, 3. After that, the user just has to Confirm the Selected account, and the payment will be initiated in the account that was selected, represented by “Através da Conta:”, as seen in 4.

Finally, when the user confirms the payment, 5, the message to add the contact will not be shown again, since it is not a new contact, 6.

3.2 Solution Architecture

In this section, Figure 3.7 represents the solution architecture to the objective previously mentioned in section 1.2. This section will include the solution for each objective that was presented and explain each of the decisions that were made in order to reach the final architecture.

The first phase was to decide how was the chatbot going to be built and how could it be easily integrated into different channels. I decided to use of chatbot platforms that can speed up and simplify
the development.

Afterward, how the chatbot will be able to process Natural Language Processing (NLP), to understand
the users’ requests, in a free speech way, not forgetting that it should work for Portuguese.

When the chatbot understands the user intent, the chatbot should be able to safely access the
BankOnBox services, which will enable the user to perform banking operations, as a matter of fact, the
bot is the one who will make the banking actions for the user.

The final objective was to enable the user to speak with the bot, for that it will be used the speech
recognition tool, which has to be able to understand Portuguese.

Throughout development, some features were implemented to improve the final solution.

3.3 Platform Choices and Components

I chose to develop the chatbot using the Microsoft Bot Framework, the main reasons are that it makes
it easy to integrate with their LUIS NLP service, which supports the required language, Portuguese, the
NLP is one of the best in the market [38], it has easy to understand documentation, and development
can be easily started. I must also say that since this is a project being developed in Link, Link works
closely with Microsoft, I had easier access to the resources in Azure, Azure is Microsoft’s Cloud Platform,
that contains all the software needed to develop the chatbot.

3.3.1 Microsoft Azure

Microsoft Azure is Microsoft’s cloud computing platform, providing a wide variety of services you can use
without purchasing and provisioning your own hardware [39]. Azure enables the rapid development of
solutions and provides the resources to accomplish tasks that may not be feasible in an on-premises en-
vironment. Azure’s computing, storage, network, and application services allow you to focus on building
great solutions without the need to worry about how the physical infrastructure is assembled.

To make bot accessible to the world, it should be hosted on a web hosting service. There are
a number of cloud platforms, which provide hosting services. The most common cloud platforms are
Amazon Web Services, Microsoft Azure, and Google Cloud Platform. All of them have similar services
for web application hosting and the possibility to use limited functionality free of charge, but since it was
chosen the Microsoft Bot Framework, for the chatbot implementation, the best option for hosting became
Azure, which will allow to deploy hardware and utilize software provided by Microsoft.

3.3.2 Microsoft Bot Framework

Microsoft prides itself on releasing the first true platform for text-based chat interfaces and an early start
on building bots that resonate with people on an emotional level. It introduced the chatbot Xiaoice in
China in 2014 and the chatbot Rinna in Japan in 2015. They have been successful not just as tools for
people getting basic information, but also as entities people enjoy interacting with at an emotional level.
Millions of people use these two chatbots for emotional support, to the extent that 25 percent of Xiaoice
users have told the bot, “I love you.”.
Microsoft introduced its own bot framework in early 2016. Microsoft bot framework SDK, similarly to other frameworks, provides the resources a developer needs to build an intelligent conversational chatbot that can provide a natural interaction.

The main components of the framework are presented in the following sections.

3.3.2.1 Bot Connector

Bot Connector is a service, which allows a bot to exchange messages with any of the channels that are available in Microsoft Bot Framework, like Facebook Messenger, by using REST API and JSON format messages over secure protocol HTTPS. This also enables the option to communicate with any client’s personal chat application by using the Direct Line API. The Direct Line Channel allows the developer to connect to their bot from anywhere. When you connect a bot to other channels, say Facebook, Skype, Cortana, Slack, etc. you are ‘locked in’ so to speak in developing specifically to accommodate those clients.

When a user sends a message from any of the available channels, when the Bot Connector receives the POST request to the endpoint URL that is specified during bot registration, the request contains an Activity Object, which defines a message that is exchanged between bot and user, the most common activity type is ‘message’, but there is also ‘typing’, and ‘event’, which can determine the behavior of the chatbot. Basically, an activity is nothing more than a JSON format message [40], which itself is a compact, human-readable, open-standard format, simple and fast data exchange (parsing) between systems. Listing 3.1 is an example of this Activity message.

```
{
   "type": "message",
   "text": "tell me my balance",
   "id": "bf3cc9a2f5de...",
   "timestamp": "2016−10−19T20:17:52.2891902Z",
   "channelId": "channel's name/id",
   "from": {
      "id": "1234abcd",
      "name": "user’s name"
   },
   "conversation": {
      "id": "abcd1234",
      "name": "conversation’s name"
   },
   "recipient": {
      "id": "12345678",
      "name": "bot’s name"
   },
   "serviceUrl": "https://smba.trafficmanager.net/apis"
}
```

Listing 3.1: User’s Activity Message Example

As explained previously, Listing 3.1 is the representation of an Activity, on the very first line you can
see the activity type, ‘message’, so the bot would understand that the user is trying to communicate with it, and that there is what the user actually wrote (or said), in the field ‘text’, where the user asks for his balance, and finally at the very bottom, the ‘serviceUrl’, which is the endpoint URL to where the chatbot should send its response to. The other fields provide context to the bot, where the message comes from, from who and to whom, which can be consumed by the bot, for example, to store user conversation history.

When your bot sends a request to the Connector service, this service being the channel from which the user is interacting with the bot, it must include information that the Connector service can use to verify that it is communicating with the right bot. Likewise, when the Connector service sends a request to your bot, it must include information that the bot can use to verify that the channel is allowed to communicate with the bot, the credentials, in this case, the chatbot’s application access token.

This is basically done by sending an HTTP POST request, the POST method requests that a web server accepts the data enclosed in the body of the request message, but specifying the access token, in the ‘Authorization’ header of the request, using the format shown below in Listing 3.2:

```
POST https://lasttestlinkbot.azurewebsites.net/apis/v3/conversations/12345/activities
Authorization: Bearer ACCESS_TOKEN
#(JSON-serialized Activity message goes here)
```

Listing 3.2: HTTP POST request example

### 3.3.2.2 Bot Builder

The Bot Builder is an SDK for .NET Framework developers for developing bots using Visual Studio and Windows. The SDK supports C# and Node.js. The kit consists of Bot Application, Bot Controller, and Bot Dialog templates. Bot Application templates already contain a simple project with all of the components for a bot already integrated with cognitive services. These already include methods to accept messages and a dialog builder to generate an appropriate response.

### 3.4 NLP - Microsoft Language Understanding (LUIS)

LUIS stands for Language Understanding Intelligent Service, an entirely cloud-based machine learning-based service to build natural language into apps, in this case, a chatbot. LUIS is built on prior work in Microsoft Research on interactive learning [41]. Since Microsoft Bot Framework was chosen as our bot service, and it even includes support for Portuguese language, LUIS was an obvious choice. Bot Builder SDK, section 3.3.2.2, even provides methods to integrate cognitive services like LUIS.

LUIS brings in artificial intelligence (AI) to applications so that computers and humans can speak with each other seamlessly, it allows for developers to create “smart” applications, without having to think or worry about designing the various Natural Language Processing (NLP) and Machine Learning (ML) techniques, developers can build custom natural language understanding (NLU) models interactively,
with the ability to improve models based on real traffic using advanced ML techniques. LUIS technolo-
gies capitalize on the continuous innovation of Microsoft in Artificial Intelligence and its applications to
natural language understanding with research, science, and engineering efforts dating back at least 20
years or more.

LUIS was designed to identify valuable information in conversations, LUIS interprets user goals (in-
tents) Intents are the crux of conversational UI in chatbots. The intents represent what the users are
looking to accomplish, like in the previous example in both phrases the user was trying to check “Account
Balance”.

LUIS can be defined in three key concepts:

- **Utterances**: An utterance is text input from the user that your app needs to understand. It may be
  a full sentence or a fragment of a sentence, basically, you could see it as a request or a command.
  Utterances are not always well-formed, and there can be many utterance variations. Ex: “Give me
  the balance from my savings account”.

- **Intents**: An intent represents actions the user wants to perform. The intent is a purpose or goal
  expressed in a user's input. The intents represent distinct commands that your language under-
  standing service differentiates, you could also think of this in terms of a function, especially after
  seeing how a chatbot routes a given query, section 4.6.2, each intent should represent a different
  function to be executed. Ex: "Give me the balance from my savings account". - could be defined
  as intent Balance.

- **Entities**: An entity represents detailed information that is relevant in the utterance, extraneous
  information about the intent, the parameters that may be necessary to fulfill a request. They can
  also be data you want to pull from the utterance. Entities are optional but highly recommended.
  Ex: “Give me the balance from my savings account” - "savings account" could be defined as an
  entity, it is the account from which the user wants the balance.

  Entities can be extracted with machine-learning, which allows LUIS to continue learning about
  how the entity appears in the utterance, they can be extracted without machine-learning, match-
  ing either exact text or a regular expression, or even in patterns can be extracted with a mixed
  implementation, these are the types of entity LUIS offers:

  - **Simple entity**: A simple entity describes a single concept.
  - **Hierarchical entity**: A hierarchical entity represents a category and its members.
  - **Composite entity**: A composite entity is made up of other entities that form parts of a whole.
  - **List entity**: A List entity is an explicitly specified list of values. Each value consists of one or
    more synonyms.
  - **Pattern.any**: A variable-length placeholder used only in a pattern's template utterance to
    mark where the entity begins and ends.
  - **Regular Expression, Regex**: A regular expression is best for raw utterance text. It ignores
    case and ignores cultural variant.
  - **Prebuilt entity**: LUIS provides prebuilt entities for common types like Number, which can be
    used for the amount of money you want to send in a payment operation.
To train the chatbot engine, entities that are expected to give the same actions are typically grouped together. Taking the previous example again, the entity “Savings Account” may be trained to recognize “savings account”, “savings”, “saved”, “money saved”, this way, the engine is able to identify the context even though said in such a different way. For LUIS to be able to interpret user’s intent and context it must be trained, the developer has to create the type of instances the system should be expecting and can do something with, and then it should provide it with a set of examples strings that exemplify that intent. The more examples the systems have for each query the more accurate the system gets. Also, every message sent to LUIS can be reviewed, in order to assure the system made the correct intent prediction, the developer can then correct the prediction, in order for the next time the same or similar message arrives it will be correctly tagged, or also confirm that the prediction was correct, either way, you can select to train the model with every new message sent by users and continuously upgrading the system. LUIS receives text (the “question”) as input and produces a prediction response that is sent to the chatbot. This response contains the scoring for the intents, being the most relevant the top intent, and zero or more entities. The output message does not contain the “answer” to read back, rather it is classifying the input into categories that the chatbot can analyze. This way the chatbot can then use the top scoring intent of the input (rather than the raw input text) to decide what to do.

There are always drawbacks of depending on an external system and service, our application fate is decided by a third party, what I mean is, our application will render useless if the service is down, under maintenance or critically changed. Not only that but LUIS also has some limitations, it only provides 1000 free endpoint hits a month. If the developer wishes to use it more extensively, it has to pay a monthly fee. There is also a set limit for the number of intent and entities the developer can specify, an application may have at most 80 intents and 30 entities, which in this project’s case, or at least during its development, will not really be a problem, but it is something to keep in mind.

### 3.5 Speech Recognition - Bing Speech API

The chatbot should also be able to recognize user speech, and speech recognition had to be able to understand the Portuguese-Portugal language. There weren’t that many options for Portuguese speech recognition, fortunately, Microsoft’s Bing Speech API is one of them.

Microsoft’s Cognitive Services Bing Speech API, which converts spoken audio to text. The API can be directed to turn on and recognize audio coming from the microphone in real-time, recognize audio coming from a different real-time audio source, or even recognize audio from within a file. The API uses the same speech-to-text and text-to-speech services used in Cortana. Also if the application needs to “talk” back to their users, this API can be used to convert text that is generated by the app into audio that can be played back to the user. The Text-To-Speech API enables you to build smart apps that can speak, although in this solution I do not intend to use this option, because even though it works well, it still feels like talking to a robot, it just does not feel natural.

The Bing Speech API provides two ways for developers to add Speech to their apps: REST APIs or WebSocket-based client libraries [42].
- REST APIs: Developers can use HTTP calls from their apps to the service for speech recognition.

- Client libraries: For advanced features, developers can download Microsoft Speech client libraries, and link them into their apps. The client libraries are available on various operating systems, using different programming languages. Unlike the REST APIs, the client libraries utilize WebSocket-based protocol.

The choice of this one is important since they provide different usage cases, I made the decision to use the JavaScript client libraries since they supported all usages. REST APIs does not support the most intuitive way for speech recognition, which is real-time, continuous stream audio recognition, this enables users to transcribe audio into text in real time, and supports to receive the intermediate results of the words that have been recognized so far, instead of having to record the audio sample, send it to the API, and wait for the API to return the recognized text. The client libraries also support integration with language understanding, using LUIS to extract intents to aid speech recognition, that the REST API does not.

3.6 Bing Spell Check API

One of the natural language understanding challenges is that the user can make spelling mistakes while writing, even tho nowadays it may be the way some users usually to talk [43], but instead of only training the bot to expect some of these errors, and to aid ease LUIS’ correct prediction, the Bing Spell Check API was used, to automatically correct user writing errors.

The Bing Spell Check API is part of Microsoft’s Cognitive Services, which allows you to perform contextual grammar and spell checking in an utterance. Instead of relying on dictionary-based rule sets, which means comparing each word with a known list of correctly spelled words, i.e. a dictionary. Bing spell-checker leverages machine learning and statistical machine translation to provide accurate and contextualized corrections, which is better compared to the rule-based approaches because, Rule-based systems require the manual development of linguistic rules and gigantic bilingual dictionaries for each language pair, which can be costly, while Machine learning is based on probabilities and statistical data model, which can be built much quicker, but require a big corpora of examples. Basically, machine learning is based on training, instead of programming strict rules, to dynamically train a constantly evolving and highly contextual algorithm. The spell-checker is based on a massive body of web searches and documents.

Bing Spell Check API is supposed to have the following features:

- **Multiple spell check modes**: Perform contextual grammar and spell checking.
- **Slang recognition**: Recognize common expressions and informal terms in the text, for example, “Moola/Moolah” and “Dough” can be used to describe money, while in Portuguese you have the word “guita”.

  Portuguese: “Quanta **guita** tenho?”
  English : “How much **dough** do I have?”
• **Homophones correction**: This corrects the usage between words that sound similar but differ in meaning, for example, in Portuguese, "cem" and "sem", which corresponds to, "a hundred" and "without", which would not really be a spelling error but could be, depending on the context:

   Portuguese: "Quero transferir sem euros para a Alice."
   English: "I want to Transfer without euros to Alice."

   would be corrected to

   Portuguese: "Quero transferir cem euros para a Alice."
   English: "I want to Transfer a hundred euros to Alice."

The Bing Spell Check API offers suggestions for misspelled words, and contextual spell checking for the text provided by a third-party application. This API uses JSON format for data exchange and API Keys for authentication. This API will be integrated with the LUIS app, from section 3.4, but it was only able to correct misspelled words in utterances before LUIS predicts the score and entities of the utterance.

### 3.7 Facebook Login & Facebook Graph API

To provide the user with a different possibility to authenticate itself, other than the BOB’s authentication method and still having access to BOB’s operations, the chatbot will allow the user to authenticate itself with its Facebook credentials, which will link its Facebook account with the corresponding BOB account.

Facebook Login is a fast and convenient way for people to create an account, or in this project’s case, link an existing account with the user’s Facebook account, and from then on, to log into the application.

Facebook Login is available in a wide variety of platforms, which obviously includes Web applications, and it enables two scenarios, authorization, using the OAuth protocol, section 2.5.1.1, for recognition and confirmation of a user’s authenticity, and asking for permissions to the user’s public profile information. The application can also ask for additional read and write permissions. Read Permissions allow you to read all other information that someone posted to their Facebook profile, while write permissions give permission for your app to post content on the user's behalf.

Apps may ask for the following two permissions from any person without submitting for review by Facebook:

- Public profile: id, first_name, last_name, middle_name, name, name_format, picture, short_name,
- email (which should not be used for spamming the user)

To ask for any other permission, your app will need to be reviewed by Facebook, before users can log in into your app with Facebook. Facebook Login provides us with the Access Token, which does not actually identify the user, it serves as authorization to access the user’s data and can be used by the app to make Graph API calls, in order to obtain a unique identification of a Facebook user.

Facebook Graph API [44] is the primary way to get data into and out of the Facebook platform, it is an HTTP-based API that can be used to make you application query data and utilize user’s data.

For this project, the chatbot will only be retrieving and using the user's public id, the user's unique identifier number on Facebook, that does not personally identify you but does connect to your Facebook profile. But, the API can be used to allow an application to programmatically do many different operations...
on the Facebook platform, it can be used to query different profile data (email, friend list, birthday, etc.), manage your Facebook pages, upload Photos/Videos, make/delete a facebook post, depending on the permissions given by the user.

The application will need an access token, which was explained before, obtained via Facebook Login, in order to make Graph API calls from your app. Graph API requests go through a host URL, most usually "graph.facebook.com", and the object ID of the node you are trying to access.

### 3.8 Azure Cosmos Database

For the chatbot to be able to identify different users, to know whom it is communicating with and be able to perform operations for the corresponding user, without having to ask the user to re-authenticate itself for each operation, a storage system was also included, in order to register users, to identify them and to store and retrieve any user information, as necessary.

Azure’s Cosmos Database, Cosmos DB [45], is Microsoft’s proprietary globally-distributed, multi-model database service. A database is an organized collection of information that can be easily accessed, managed and updated, using database services like Cosmos Db, a system is able to interact with end-users, other applications, and the database itself to capture, store and analyze data.

You can use Cosmos DB to quickly create and query key/value databases, document databases, and graph databases, all of which benefit from the global distribution and horizontal scale capabilities at the core of Azure Cosmos DB.

Azure Cosmos DB guarantees less than 10-ms latencies on reads and writes at the 99th percentile [46], this capability allows for a very high volume of data messages with minimal delay and fast queries for highly responsive apps. Cosmos DB secures users information with enterprise-grade security and compliance and is the first and only service to offer industry-leading comprehensive SLAs for 99.999% high availability [47].

Since you only pay for the throughput and storage you need, the system can be scaled with the arrival of new clients. Azure Cosmos DB allows you to independently and elastically scale storage and throughput at any time, so the system can accommodate as many users as needed.

With Cosmos DB the chatbot will be able to store users’ account data, these accounts are defined in section 4.3, but it will allow both login services, the login through the BOB API’s authentication method, and Facebook’s login.
Chapter 4

Implementation

This Chapter will focus on explaining, regarding the Solution that was defined in the previous Chapter 3, the architecture implementation, how was the bot implemented and integrated with the other components, the decisions made during the implementation and the problems that were faced along the way.

The implementation explanation will start by focusing on the Natural Language Processing implementation, LUIS 4.1, and the Chatbot capability to perform the user’s requests, but all bot components were equally important in order to reach the final application.

The initial idea was to focus on implementing the bot locally, and this would allow us to easily integrate the Chabot in different Channels (WebChat, Facebook, Skype, etc), but, as I went to a deeper level of the study of the application, and how it could be integrated into each channel, I started realizing that this option would not be as easy as one would think, because, some channels were designed, and built, differently, and there’s no way to really debug the channels applications in real time, the only way to debug a bot is locally through the use of the Bot Framework Emulator and debugging using an integrated development environment (IDE), in this case, Visual Studio [48]. When testing the bot hosted in Azure, the only solution to debug it, to see what was wrong with application, would be to send the results as a message to the chat window, just to see what was wrong. Even though I set up Continuous Deployment [49], which as the name suggests, allowed me to be able to easily deploy my code in the Azure Services VM to automatically released into the production environment, making changes available to me and to public user, every time I had an error, I would have to go to my local code, make changes, compile it, commit & push it to Github [50], and then wait for Azure to be in Sync with Github and deploy the new code, it was tiring and time consuming.
4.1 LUIS, Natural Language Processing

To start using LUIS, developers begin by going to LUIS platform [51], creating a new LUIS “application”, if the steps for creating a bot with the LUIS template are followed [52], the LUIS app was automatically created for you. After that, specify the intents and entities needed in their domain. Thinking about the intents that are important to your application’s task, for the purpose of this project I had to make sure that the bot had at least one intent for each of the BOB API features, these were the intents initially needed and that will be the focus on this section:

- Account Balance (Intent: Saldo)
- Initiate Transfers (Intent: Transferencias)
- Transfers Information (Intent: Consultas)

LUIS implementation should start by using only as many intents as you need to perform the functions of your app. If you define too many intents, it becomes harder for LUIS to classify utterances correctly. Although, if you define too few, they may be so general as to be overlapping.

To make LUIS able to correctly identify these intents, and in order to train it, LUIS needs to be fed with utterances examples that should represent a possible user request for the desired intent. Microsoft provides a graphical web interface for creating and training the application. When the new application is created there is one intent present, namely the None intent, that is explained in section 4.1.4, but then it also has the option to create your own intents [53].

4.1.1 Saldo Intent

Figure 4.1 shows the intent “Saldo”, and it’s corresponding fed utterances, as you can see, for each utterance LUIS associates a corresponding score, this score represents LUIS degree of certainty, that the utterance is associated with that given intent. This score is greatly increased the more utterances you have, ideally, you want your intents to be distinguishable from one another with a high degree of certainty, that is, to have one intent with a very high confidence score, while other intents have lower scores, if you revisit the utterance list, from figure 4.1, you can see that, for some utterances, LUIS is so confident on its prediction that it scored them with a score of 1.000, i.e. LUIS is 100% certain that the utterance belongs to that intent, this is usually caused by

LUIS does not just score the intent that was predicted, what it does is score the utterance according to each of the intent you have created, and chooses the highest value as the intent prediction, which can mean that some times, even the top scoring intent, may have a low score, meaning that it is important to decide whether to accept the intent predicted was a valid one, if the score is too low, it might be for one of these reasons, either the prediction really is wrong, and you should not consider it, or the prediction was actually right, but something threw LUIS off balance, sometimes when the same words or expressions that would normally appear on other intents are used in other intents utterances, LUIS can get a little confused, but it is one of those cases where “training makes perfect”.

So, to drastically improve the quality of LUIS predictions, you can specify entities. Entities help in identifying the parameters which are required to take a specific action. While intents are required,
Figure 4.1: Saldo Intent, Utterances and Score Examples

Figure 4.2: Saldo Entities, Labeled vs Unlabeled
entities are optional. You do not need to create entities for every concept in your app, but only for those required for the client application to take action. If your utterances do not have details your bot needs to identify, you do not need to add them, but from this experience, I would say, they are the key for correctly identifying user Intent, not just additional actions. The entity represents a word or phrase inside the utterance that you want to be extracted. An entity represents a class including a collection of similar objects. Let’s continue with the same example, the intent “Saldo”, if the user wants to check its balance, it would probably write a message that would have a word that would relate to the balance, in Portuguese, “saldo”, or at least that’s what you can expect right?

So I started by creating an entity named “saldo” that is responsible for any word or expression that could be synonyms, or that could mean the same thing as the balance. Once a machine-learned entity is created, you need to mark that entity in all the example utterance of all the intents it is in, like the basic expression “how much money do I have?” (in Portuguese, “quanto dinheiro tenho”), the intent should be the same, but the way of saying it is completely different but LUIS if I define the whole phrase as “saldo”, LUIS will not even know the difference, label all the synonyms or different expressions that are normally used as “saldo”, LUIS will start to automatically label them as it finds them in new utterances, figure 4.2 shows some “Saldo” intent utterances, and how their entities were labeled, which are represented in blue. To the left are, basically, how LUIS sees the utterances, and to the left we have the real utterance sentence, showing some different ways that LUIS is able to identify that the user is asking for its account balance.

The objective is to prepare the bot with all the different possibilities the user could say the same thing, to not be so restricted with specific keywords and being able to speak as they normally would, or want to, because users tend to text differently then they actually speak, especially when trying to use a service, users may, but tend to not really wanting to have a conversation, but to get this done quickly, that is why it is important to try and anticipate who and how will our application be used.

From this point on, every time you send a new utterance for LUIS to predict, LUIS uses the trained model to propose entity labels, as you can see in figure 4.1.

In a similar way that was explained in this section, the intents “Transferencias” e “Consultas” and corresponding entities, “transferir” e “consultar”, were created, in order to be able to implement the required operations of BOB’s API.

### 4.1.2 Transferencia Intent

Having the basic “Transferência” Intent created, I started working on additional actions for the user. If this project was meant to be just to do the basic operations, it might as well just had been a click of a button, one for each operation, but the user is supposed to not only be able to call this functions but also do more specific requests, let's take for example the “Transferência” intent, where for BOB API Transfer Request, we have to send an IBAN number and the amount of money to make the transfer request, so the user should be able to define all these components with just one command.

Entities can also be used to understand additional user intent, in this case, we want to be able to identify an IBAN number and the amount to money to transfer. There are many types of entities, as
explained in section 3.4, we tried to take advantage of this, for example, since we know an IBAN as a specific size and model, 25 digits and starts with the letter PT (for Portugal), and after experimenting with simple entity, I decided to create a Regular Expression Entity for the IBAN, because regular expression matching is applied, this means that every word with the format I choose would be labeled automatically, since we have the same 2 letters in the beginning, and then 23 random numbers from 0 to 9, we get the following expression format, PT\{0-9\}\{23\}, to identify the IBAN.

On the other hand, the amount of money is not that simple. Simple entities can be anything, and there are basically 2 ways of writing an amount, you either write the number name in full, like “a hundred and fifty-two” or you just write the number “152”. This is one of the linguistic problems that had to be addressed. First, a simple entity named “dinheiro” was created, the entity worked for numbers, but when writing the full name of the numbers, it was not identifying new numbers, because each number was different for LUIS. Fortunately, LUIS updated their prebuilt entities, from section 3.4, to work in Portuguese, which was tried previously without success, but with this I was able to use both to my advantage, I use the “number” prebuilt entity to implement the full written numbers, which automatically turns the full written number directly to its integer value, and this was the way I was able to predict the amount of money a user wants to transfer.

To finalize the “Transfêrencia” Intent, we tough that the user should not have to write the IBAN every time it wants to make a transfer, especially if it regularly uses the same IBAN, so the chatbot will be responsible for keeping user IBAN contacts, but LUIS as to be able to identify names, the problem is there are a lot of names, at first I thought I would have to introduce an utterance for every single one name, in order for LUIS to be able to predict the transfer recipient, but in order to not having to do individual training for each name, there's an option to create a Phrase List, from section 3.4, which basically feeds a simple entity with entity examples, without having to create utterances and label them. I downloaded a list with all the Portuguese official name list [54] and linked it with the new “destinatario” entity, which is done by giving the same name of the entity to the corresponding Phrase List. Unfortunately, the whole list could not be imported, since Phrase List is limited to 5000 entries, and the list is over 7000, so there are still some names that would not be recognized but can be trained to, but it saved a lot of time, and it's efficiency at labeling them correctly.

4.1.3 Consulta

The “Consultas” Intent should describe the intent of wanting to see some information on your banking movements. In this case, it was required that the user should be able to select the date of the movements it wants to see. To do that, a composite entity “Data”, referenced in section 3.4, was created, which should be composed by child entities, other simple entities. In this case, the “Data” entity has three different child entities the day, “dia”, the month, “mês”, and the year, “ano”. I picked using a Composite entity because it proved to help to identify the three entities together, I had already defined the 3 simple entities, but there was always a problem when trying to label all three at the same time, either one of them would not be correctly labeled, or there would be confusion between the day and the month, they overlapped sometimes, the same word was labeled with two different entities. Composite Entities should
be used when you expect certain entities to be next to each other, in this case, we can assume that the
date will always be next to each other, they may be separated by different thing, but with composite
date, as long as they are next to each other, without any non child entities in the middle, Composite
ees are focused on trying to find their Child Entities, and with this I could identify dates in different ways, like
“DD/MM/YYYY” and “DD of MM of YYYY”, but also, composite entities does not require all child entities
to be present for it to be labeled, therefore, other combinations were possible like day and month and
month and year, or even just one of the three.

Also, I had to have in mind that the month could be a word and not only a number, but this part was
also done with training. There are not that many months nor writing variations, it is either the full name
or the abbreviation, for example, January can be just “Jan”.

Finally, a user is able and its normal to have more than one account, therefore it should be able to
choose with which account its request should be performed, Simple entity “contaBancaria” was created,
but it’s probably one of the most unreliable ones, first the user can have any name it wants for its account
name, so I can train to work with the accounts I’m given, but for other accounts I created Patterns for it.
Patterns allow an intent to expect certain phrases with or without entities on them, A pattern is matched
based on detecting the entities inside the pattern first, then validating the rest of the words and word
order of the pattern. Entities are required in the pattern for a pattern to match. The pattern is applied
at the token level, not the character level. In this case when wanting to define an account for “Consulta”
the intent is expecting the following pattern:

\{
Consultar
\}
conta
\{
contaBancaria
\}

where \{
Consultar\} e \{contaBancaria\} represent entities.

The pattern aids LUIS identifying the word after “conta” should be expected to be an entity of type
“contaBancaria”.

But that is not all Patterns are for, Patterns were designed to improve accuracy when several utter-
ances are very similar, in order to help LUIS recognize intents, patterns greatly diminish the necessity
to train an intent with so many utterances, if LUIS can correctly identify the entities, he can match those
entities with the created patterns, to greatly increase its intent prediction, but a pattern is not a guarantee
of the intent prediction but it is a strong signal, so, I went and started applying Patterns to every intent,
with the corresponding and necessary entities.

This entity was later also applied to the “Saldo” Intent 4.1.1 with the same purpose and usage. At the
start of the implementation, there was only one account available in the provided BOB Account, which
changed during implementation, now as you could see seen in 4.10, the user has three accounts from
which it can choose.

4.1.4 None Intent

The presented intents were, in my opinion, the most important NLP cases for the project, except for the
None intent, which sometimes is forgotten but very important, the None intent is important to every app
and should not have zero utterances, [55]

The None intent is a catch-all or falls back intent. It is used to teach LUIS utterances that are not
important in the app domain. When an utterance is predicted as the None intent and returned to the bot
with that prediction, the bot can ask more questions or provide a menu to direct the user to valid choices in the bot, or in our case, just say that it can not help the user with what he just said. No utterances in None intent tends to mess with LUIS predictions. If you do not add any utterances for the None intent, LUIS is forced to predict an utterance that is outside the domain into one of the domain intents. This will mess with the prediction scores because you have no way of teaching LUIS to invalidate predicting the intent for those utterances.

### 4.1.5 Luis Training

Training is the process of teaching your Language Understanding (LUIS) app to improve its natural language understanding. LUIS success is based on training, the more you utterances you provide it with, the better results you will have, the more options you are covering, the better it will react, and it should be always learning more, the more you and the user use it. When you train a LUIS app, LUIS generalizes from the examples you have labeled and learns to recognize the relevant intents and entities in the future, which improves its classification accuracy.

LUIS has basically two ways of being trained. The first one, the obvious one is to add utterances to each of your intents and then it is just a single step of pressing the Train button, which can be seen in at the top right of figure 4.1, with a green light, meaning there are no changes to be trained, otherwise it would appear red.

It might sound obvious but whenever you add a phrase list feature, a pattern feature, an entity, whatever it is, the first thing you should do right after is re-train your model. Then, add a few utterances and map entity values with their corresponding types and re-train again. Only a few examples are enough for LUIS to automatically detect entities correctly with the new utterances.

Training and testing an app is an iterative process. After you train your LUIS app, you test it with sample utterances to see if the intents and entities are recognized correctly. If they are not, make updates to the LUIS app, train, and test again.

The other option and I think the most important is that LUIS allows review of utterances sent to its endpoint, meaning every new user message that was predicted, can be Reviewed, meaning that you can view what LUIS predicted the utterance as, and all the entities, giving you a chance to edit it, so that in case a wrong prediction was made you have a change LUIS prediction, so for every mistake it makes, you can make sure it will not happen again. This is what will help satisfy your users, the more different users you have using your application, the more different examples you will have, the better your prediction model will be.
4.2 Authentication and Authorization

In this section, it will be explained how the two different methods that the user has to authenticate itself were implemented, first BOB’s authentication and authorization method, through OpenId Connect, and for the second authentication method, Facebook Login.

4.2.1 OpenID

As previously mentioned, in section 2.5.1, the main Authentication and Authorization mechanisms for the BOB API, is done through OpenID Connect, from section 2.5.1.2.

There are many ongoing implementations of OpenID Connect, but the OpenID Foundation keeps track and certifies any libraries that have correctly implemented OIDC, these provide an incredible help when developers want to implement their own Authorization Servers, everything from security, connectivity, integration, etc, are already precoded, ready to use, all you need is to take advantage of the functions of these libraries. These libraries have been developed in different programming languages. The idea behind these libraries is that the developers who want to build an OpenID Provider or a Relying Party, will get the library for the corresponding programming language and then customize accordingly. Since the project is being developed in C#, using .NET Framework, I decided to take advantage of the IdentityModel.OidcClient2 library [56]. IdentityModel.OidcClient2 is a C#NetStandard OpenID Connect Client Library for native Applications, an implementation of the OIDC/OAuth2 for native apps specification for C#.

Following the library documentation, and referring to the BOB API login requirements, that can be found in [57], we created a OpenID client with the client options, shown in listing 4.1 , required for authorization:

```csharp
var options = new OidcClientOptions
{
  Authority = AUTHORITY_URL,
  ClientId = CLIENT_ID,
  ClientSecret = CLIENT_SECRET,
  RedirectUri = BOB_REDIRECT_URL,
  // examples
  // "http://lasttestlinkbot.azurewebsites.net/signin-oidc",
  // "http://localhost:5002/signin-oidc",
  Scope = "openid ais.consacct ais.consacctbal ais.consaccttrans ais.consacctlist pis.
           initpay user.name offline_access",
};
var client = new OidcClient(options);
```

Listing 4.1: OpenID client Options and Object creation

- **Authority**: The URL of the OpenID provider.
- **ClientId**: The unique public application identifier as registered with the provider.
- **ClientSecret**: This is a secret known only to the application and the authorization server, it is then used in conjunction with ClientId, towards the OpenID Endpoint, it is validated and afterward
permitted to proceed with the OIDC protocol.

- **RedirectUri**: The URI that should the Authentication Code response should be sent, after login success and permissions consent given by the user. We provide 2 examples one was the URL used while working in a local environment, and the other is the Web Application URL.

- **Scopes**: The allowed scopes of access that the Application is given consent to. This list of scopes specifically states what particular access the application will have. In this case, we are prompting the user for all available permissions, which they can accept or decline. For example, the "offline_access" is referring Regarding Refresh Tokens. If set to true, then a Refresh Token is sent during the exchange of the Authorization Code for the Tokens. Otherwise, no Refresh Tokens are issued for this Client.

If any of this option does not match with the requirements implemented in BOB's IdentityServer, when the user tries to log in, an unauthorized Client error will be shown in the BOB Log in the page.

BotFramework constrains opening URLs to a click of a button, this is a safety measure to ensure that bots don’t take user unwillingly to another website, which could be malicious. A bot is only supposed to send content back to the user after a message is received. To open a URL, user-interaction is required. This is why you have to use the Card-actions, in this case, OpenUrl action, which when invoked, show the given Url by launching it in an external web browser. To create a valid login request Url we execute the function PrepareLoginAsync, from Listing 4.2, creating the necessary State Data, which include the state, nonce and the start Url.

```csh
// generate start URL, state, nonce, code challenge
var state = await client.PrepareLoginAsync();
```

Listing 4.2: OpenID client function, PrepareLoginAsync

The state parameter [58] protects the end user from cross site request forgery attacks (CSRF) [59]. Once authorization has been obtained, the authorization server redirects the user back to the client with the required binding value contained in the "state" parameter. The binding value enables the client to verify the validity of the request by matching the binding value to the user saved authenticated state.

The nounce serves as a token validation parameter [59], to mitigate replay attacks. The value is passed through unmodified from the Authentication Request to the ID Token. If present in the ID Token, Clients MUST verify that the nonce Claim Value is equal to the value of the nonce parameter sent in the Authentication Request.

We retrieve BOB's Login URL, the Identity Server URL, from the State Data and insert it in the Bob Login card action, which is a button that opens the picked URL.

In order to know to whom each log in request belongs to, we retrieve the user conversational state, Listing 4.3, which will allows the bot to not only know who the request belongs to, defined by “User”, but also to know where to find the user, what “Channel” is the user communicating from (emulator, webchat, skype, etc...), and what's the conversation reference, so that the bot can warn the client that the log in was successful and save the log in information for the right client.
ConversationReference conRef =
        new ConversationReference
        {
            ActivityId = message.Id,
            Bot = new ChannelAccount { Id = message.Recipient.Id, Name = message.Recipient.Name },
            ChannelId = message.ChannelId,
            User = new ChannelAccount { Id = message.From.Id, Name = message.From.Name },
            ServiceUrl = message.ServiceUrl
        };

Listing 4.3: Chatbot Conversation Reference

The conversation Reference is stored along with state data, in Cosmos DB database, state collection, section 4.3.3, using the state value as the unique id identifier for the database, since it will ensure that the request received is a valid one, if the state received in the Log in response is not present in the Database, it’s not even worth handling the request, we know something is wrong with it.

To deal with the server login response, an HTTP handler was defined, in order to route HTTP requests to the right handler, the handler should be defined on the application configuration file, “.config” file, by adding the code from Listing 4.4.

The <handlers> element [60] defines the handlers registered for a specific URL name extension, in this case, we define the path attribute, that defines the name extension where handler mapping applies, as “signin-oidc”, this extension should match the one used for the RedirectURI OpenID client options, Listing 4.1, while the “type” attribute specifies the namespace path of the handler class. In summary, every HTTP request done to the host Url, with extension “{botHostedURL}/signin-oidc”, will be redirected to the OauthHandler class file.

Listing 4.4: Defining OpenID handler route .config

The OauthHandler class implements the HttpTaskAsyncHandler class, which is responsible for handling the server response. We start by retrieving the expected data from the request, including the state, which we are expecting to be present, otherwise, we ignore the request, as it already does not meet the necessary requirements. If the response data, that contains the state, was retrieved from the HTTP request, state value is used to search the database state collection, in order to ensure that the request is valid, and if there’s a user behind the request. If state Data, created with Listing 4.2 and User Conver-
sation Reference, from Listing 4.3, are successfully retrieved, the application proceeds to the last step of the OpenId flow, processing the authorize response, shown next in Listing 4.5.

```csharp
var result = await client.ProcessResponseAsync(data, state);
```

Listing 4.5: OpenID Client, ProcessResponseAsync function

The function requires the `data` retrieved from the current handler HttpContext request, and the `state` data, that was saved in and retrieved from, the database state collection. This function is responsible for the state and nonce verification, which if it works, returns the Log in result data, which gives us the user Claims, the user's identity informations, from which we retrieve and use the UserId, UserName, but also the AccessToken and RefreshToken, that are the necessary values for our users.

With the Log in data acquired, the application checks if the user already exists in the database account collection, section 4.3.2, if not, it creates the new user document in the account collection, if it exists, retrieves user previous stored data and copies it in the user's current data in the bot storage, section 4.3.1, which we only know and have access because of the user's conversation reference.

With this, the user is now logged in with the BOB credentials, and the bot is from now on able to make BOB API calls, with the use of the retrieved accessToken, in name of the user. Figure 4.3 represents the Authorization Code Flow of the operations that were described in this section.

![Figure 4.3: OpenID Connect Connection Flow](image-url)
4.2.2 Facebook Login

The implementation started by creating a new Facebook app, BOB Chatbot, following Microsoft’s guide [61], in order to be able to have access to an Application ID and Application Secret necessary to start using Facebook's Login [62], in our application.

The easiest and quickest way to implement Facebook Login would be with one of their official SDKs, However, I needed to implement browser-based login flow using entirely server-side code, using browser redirects, so I manually built a login flow [63]

First, we need to create the redirect URL to our Facebook application Login endpoint which will open the typical Facebook Login screen, that can actually be seen in 4.9.

```
https://www.facebook.com/v3.3/dialog/oauth?
client_id={app-id}
&redirect_uri={redirect-uri}
// https://localhost:44372/signin-facebook;
// https://lasttestlinkbot.azurewebsites.net/signin-facebook;
&state={state-param}
&auth_type = reauthenticate
```

Listing 4.6: Log in endpoint URL

This endpoint has the following required parameters:
- **client_id**: The Facebook application ID previously mentioned.
- **redirect_uri**: The URL that you want to redirect the person logging in back to. This URL will capture the response from the Login Dialog and must be set as a trusted redirect URI, in the Facebook app client Oauth Settings. We provide the examples used in this project, the important part is the “signin-facebook” extension that will be used to handle the requests 4.7
- **state**: This parameter should be used for preventing Cross-site Request Forgery and will be passed back to you, unchanged, that’s why in this case the state parameter corresponds to the user conversation reference 4.3, which, if sent unchanged, will result in a correct reference of user who made the request.
- **auth_type = reauthenticate**: This parameter as only this option, which enables the app to confirm a person’s identity even if it was verified previously. Facebook Login lets your app ask a person to re-enter their Facebook password at any time. This is used to prevent cases where a user leaves a device logged in or where a third-party hijacks someone’s session with your app, it might be an extra step that might be kinda annoying if you are used to always be logged in automatically in facebook, but it is not a risk worth taking, when handling sensitive data like banking information.

The created URL will then be used in the Login request card, Figure 4.9, which in the same way the for OpenID, when the card button is clicked it will redirect the user to the typical Facebook Login page, which will be explained in section 4.2.2, if the user successfully logs in, Facebook will then send the response to the previously defined redirect-uri. In order to handler this response, unlike the handler
solution presented in openid, instead of creating a handler, Listing 4.4, I decided to utilize a different approach, with ASP.NET Web API Routing.

ASP.NET Web API Routing is used for handling HTTP requests and searching matching action methods, and then executing the same. In ASP.NET Web API, a controller is a class that handles HTTP requests, the routing engine is used to route the request to the desired controller, in order to know which controller to invoke, the framework uses a routing table. By making a simple change to the WebApi-Config.cs file we added the new route, Listing 4.7, where we defined the routeTemplate, that represents the URL extension to route, and the controller that should be executed to handle the HTTP request. In summary, every time an HTTP request is made to the “signin-facebook” extension, it will be forwarded to the “FacebookLoginController” class.

```csharp
config.Routes.MapHttpRoute(
    name: "FacebookLogin",
    routeTemplate: "signin-facebook",
    defaults: new {
        controller = "FacebookLogin"
    }
);
```

Listing 4.7: ASP.NET Web API Routing, for facebook login

So, in response to a successful login facebook returns 2 parameters, the previous sent “state” unchanged, that will be used to reconnect with the right chat user and the “code”. The Controller is responsible for retrieving and handling both these parameters.

The “code” is an encrypted string unique to each login request because this redirect flow involves browsers being redirected to URLs in your app from the Login dialog, traffic could directly access this URL with made-up information. If the app assumes this information as valid parameters, the made-up data would be used by your app which could potentially be used for malicious purposes. The app should confirm that the person using the app is the same person that’s receiving the data, before generating an access token. In order to get this access token we must exchange the “code” for it, by making an HTTP GET request to the Facebook Graph API endpoint, section 3.7. This request is represented in Listing 4.8.

```csharp
GET https://graph.facebook.com/v3.3/oauth/access_token?
    client_id={app-id}
    &redirect_uri={redirect-uri}
    &client_secret={app-secret}
    &code={code-parameter}
```

Listing 4.8: Log in endpoint URL

- **client_id**: The Application ID, same as the previous one, in Listing 4.8
- **redirect_uri**: This URI must be the same as the original request_uri that you used when starting the OAuth login process, Listing 4.8
• **client_secret**: Your Application Secret, also mentioned in the beginning, from creating your application.

• **code**: The parameter received from the previous login success response.

If the request is successful, the response will include the access token. The application does not save this token, it will only be used to request for user’s facebook id, the unique Facebook identifier that the chatbot will use to identify the user. For that we do one last request to the Facebook API, the “/me” node is a special endpoint that translates to the user_id of the person, whose access token is currently being used to make the API calls. If you have the User access token, by using:

```
GET https://graph.facebook.com/v3.2/me?
fields=id,name
&access_token={access-token}
```

Listing 4.9: Log in endpoint URL

• **fields**: This defines the user information that you want to retrieve, the user is defined by the access_token that is provided. In this project’s case, it only needs the *id and name*, but different information can be retrieved as long as they are public profile information, as explained in section 3.7

• **access_token**: Should be the access token from the previous action, that was exchanged by the “code”.

There should be no problems here as long as the **access_token** is correct, in the response data there will only be the two requested fields, id, and name. The name is just to possibly greet the user when there is no BOB account associated, and with the id, the bot will search for a user in the account database, which is to be explained in section 4.3.2.

A sequence diagram is given in Figure 4.4, which explains the flow of control in authentication using Facebook, for the operations that were described in this section.

![Figure 4.4: OpenID Connect Connection Flow](image-url)
4.3 Cosmos DB

To implement Cosmos DB, an azure cosmos DB account must be created, following Microsoft’s directions [64], a new account and a new database were created, which now enables you to get access to a database URI and it’s access Key, those are the necessary parameters to do these communications, copy and store these values safely in the chatbot application.

4.3.1 Bot Collection

To keep context of conversational data, the bot has to have a defined storage, which by default uses Azure Table Storage [65].

While Azure Storage Tables is aimed at high capacity on a single region and storage-optimized pricing, while Azure Cosmos DB aims for high throughput, meaning you can read and write many documents with very low latency, a global distribution, which allows multiple failovers, and a pricing model focused on throughput, in summary, Cosmos DB is a better option for system scalability.

Knowing that the storage was changed to instead use Cosmos DB, which can be defined in the Chabot Application Start, by creating the Cosmos DB data store object with the created Cosmos DB URL and key access, and register the created object as shown in 4.10.

```csharp
var uri = new Uri(ConfigurationManager.AppSettings["DocumentDbUrl"]) ;
var key = ConfigurationManager.AppSettings["DocumentDbKey"] ;
var store = new DocumentDbBotDataStore(uri , key) ;

Conversation . UpdateContainer ( builder =>
{
    builder . Register ( c => store )
        . Keyed < IBotDataStore < BotData > >(AzureModule . Key , DataStore)
        . AsSelf ()
        . SingleInstance () ;

    builder . Register ( c => new CachingBotDataStore ( store ,
        CachingBotDataStoreConsistencyPolicy . ET agBasedConsistency))
        . As<IBotDataStore < BotData >>()
        . AsSelf ()
        . InstancePerLifetimeScope () ;
}) ;
```

Listing 4.10: Bot Storage configuration code

This automatically creates a new database Collection, Bot Collection, and the necessary data documents for the bot storage, every time a new user enters the conversation, the user data document, UserData, it’s created for that user, that keeps the data for the user during the conversation lifespan, on
the specified channel. Then for each conversation that is started, a conversation document is also created, that is responsible to keep the data within the current conversation on the specified channel, and finally, the private conversation document that saves data for the user within the current conversation on the specified channel.

This project’s bot only stores information in UserData, which is the document where the chatbot will save user’s session account data, in the “data” field in Figure 4.6, received from OpenId log in, from section 4.2.1, so that the bot can have quick access to any of the user information’s within the lifespan of the conversation context, and used them accordingly.

4.3.2 Account Collection

While testing the application locally in the chatbot emulator [66], every time a new conversation was started, either by refreshing the emulator a new user, with a different id, is created, meaning that every time the user opens the chat, it would have created a new User data document, like the one in figure 4.6 but without any “data” in it, so, even if the user logged in, it would lose any data that was created during its session. The bot saves some user data that is created during the session and cannot be saved in BOB’s Account, like the contact List, “listaContactos”, that is created and used only by the chatbot, to store a list of IBAN contacts. This data would be lost every time the user left the chatroom.

In order to store the user’s BOB account with the corresponding user data, and being able to retrieve this data the next time the user logged in, a new database collection was created the Account Collection. In Cosmos DB documents, the id is the document unique identifier, it’s basically the file’s name, to enable
an easier retrieval of the document of the account this “id” corresponds to BOB account unique identifier, this allows for easy document access, as soon as we get the Login Data, as explained in section 2.5.1.2, we create a new user if bob user id was not found in the database, or retrieve the data in case it exists. To do this we take advantage of the bot framework and easily create a database document client, that using the Cosmos DB needed credentials, allows easy access and predefined functions that will allows you manipulate the database as you wish, you can create, read, replace and read documents as long as you input the write parameters, as shown in Listing 4.11

```csharp
private static readonly Uri EndpointUrl = new Uri(ConfigurationManager.AppSettings["DocumentDbUrl"]) ;

private static readonly string PrimaryKey = ConfigurationManager.AppSettings["DocumentDbKey"].ToString();

client = new DocumentClient(EndpointUrl, PrimaryKey);

client.ReadDocumentAsync(UriFactory.CreateDocumentUri(databaseName, stateCollectionName, documentID));

client.CreateDocumentAsync(UriFactory.CreateDocumentCollectionUri(databaseName, stateCollectionName), documentData);

client.DeleteDocumentAsync(UriFactory.CreateDocumentUri(databaseName, stateCollectionName, documentID));

client.ReplaceDocumentAsync(UriFactory.CreateDocumentUri(databaseName, collectionName, userID), documentData);
```

Listing 4.11: BotFramework Database Client Functions

This implementation allowed me to think of how to implement Facebook Login, section 4.2.2, and link a user’s Facebook account with BOB’s account, all the bot does is retrieve the user’s Facebook unique identifier and store it, as “facebookid”, inside the same document as the corresponding user BOB account document, as shown in Figure 4.5. When retrieving the account data instead of looking up a document by its id, it was necessary to create an SQL query, Listing 4.12. This query is responsible for selecting the right document from the database, it will search every document looking for a possible document that contains the element “facebookid” that is requested, this “facebookid” is the user’s unique Facebook identifier received from Facebook Login, meaning it is unique and it will never appear in more than one file

```csharp
var facebookData = this.client.CreateDocumentQuery<AccountData>(
    UriFactory.CreateDocumentCollectionUri("botdb", "accountsCollection"),
    "SELECT * FROM Family f WHERE f.facebookid =" +facebookid + ""
    , queryOptions).ToList();
```

Listing 4.12: SQL Query, Searching Document by “facebookid”
4.3.3 State Collection

As previously explained, in OpenId section 4.2.1, there was a need to create a state Document Collection, to keep the login requests states and user’s conversation reference. This document is only supposed to be used once, so these are the only documents that are dispensable and can be deleted after the first use. Its implementation is the same as the previous account collection, as long as you provide the correct collection name and document id, an example of the document structure is shown in Figure 4.7.
4.4 User Interface

To make the bot accessible to users, the bot should be hosted on a web hosting service, in this case, when you create a bot with the Microsoft Bot Framework, from section 3.3.2, it automatically starts being hosted in Azure, but you still need to provide the user a way to get to it and interact with it.

Since the bot is being hosted in Azure, it was chosen to host a web page, as our bot user interface, so that anyone could get easy access to it. To embed the bot in the webpage [67], it needs a chat window, the message delivery platform, to integrate this, the BotFramework Web Chat was implemented [68], as our Web Chat platform, that provides the ability for users to interact with your bot directly in a web page.

Having the web page file ready and configured in the chatbot application, changes need to be made to include the BotFramework Web Chat on the page. Following the directions from its documentation [68], the HTML was changed to include the JavaScript code that invokes the Web Chat control and creating the “BotChat” application, that can be seen in Listing A.4. This open source web chat control communicates with bots by using the Direct Line API, which allowed messages to be sent back and forth between client and bot. The Direct Line is one of the channels available that allows us to create a custom client application and integrate with the Bot Framework. To allow the Web chat to communicate with the bot, it needs the Direct Line Secret Key, represented by “YOUR_DIRECT_LINE_SECRET”, in Listing A.4. This secret key can be retrieved by adding Direct Line Channel in Azure Bot Services [69].

In the end, our user interface, the chatbot, is hosted in https://lasttestlinkbot.azurewebsites.net/, and Figure 4.8 shows an example of what this interface looks like.

![Figure 4.8: Chabot Interface Usage Example](image-url)
4.5 Bing Speech API, Speech Recognition

Speech input was implemented within Direct Line WebChat [70], implemented in section 4.4.

To enable speech in the Web Chat [71], the JavaScript code that invokes the Web Chat control, was customized to include speech, this is done by enabling it via "speechOptions", which tell the bot which speech recognition and/or synthesizer to use, in this case, as referred in section 2.2, the Bing Speech API. This implementation can be seen in the HTML file code, Listing A.4, where its defined the speech recognition, "speechRecognizer", you provide it with a "subscriptionKey", the Speech API access key, that can be found by creating a Speech Service, from Azure's Cognitive Services, thought the Azure's Portal, and the "locale", which defines the language to be recognized, in this case "pt-PT" [73].

Having this, the chatbot was now able to recognize Portuguese speech, the webChat automatically shows a microphone icon, that can be seen in the bottom right of Figure 4.8, by clicking the microphone the user may be asked to allow the use of its microphone, depending on the browser and user options, but after that all the user needs is to say whatever it wants and the bot will start decoding, creating the message in real time of what the user is saying. When the application recognizes that the user stopped talking, it automatically sends the message to the user. After that everything works as if it were a text input, the message is sent via text without the bot even knowing it was done through speech recognition.

The Speech Sinthesizer was also defined, seen in Listing A.4, with the previously used subscription key, and pick the needed voice option from Microsoft’s supported languages [74], since this project is being developed in Portuguese, there was actually only one option available, and can be seen in Listing A.4, represented by the "gender" and "voiceName" parameters.

4.6 BOB ChatBot Implementation

In this section, it will be explained how the chatbot was implemented with all the previous components, to achieve the desired solution from section 3.

4.6.1 Bot Builder

To create the chatbot, Bot Services provides the core components for creating bots, including the Bot Builder SDK for developing bots and the Bot Framework for connecting bots to channels. Bot Service provides five templates you can choose from when creating your bots with support for .NET and Node.js.

When implementation started the current version of the Bot Framework was v3, during implementation Microsoft published Bot Framework v4, this caused a lot of confusion, every time I had to go and check the documentation I had to be careful to change it to v3, and some of the v3 functions where being deprecated, meaning they would stop working because they were unsafe to use but could be implemented in some other way.

Knowing that, following the steps provided by bot service v3 documentation [34], the steps are, going to the Azure portal [75], create a bot, providing the requested information, and this will automatically
create a bot for you, that is already ready to use, you can even test it right away, even though it might just repeat your message, it works.

In the previous step, while creating a bot you are asked if you want to use a template to create your bot, these bot templates are definitely one of the most useful components, they allow you to have a bot that is automatically implemented with other Azure Services, these templates provide a big boost on development type so that with just a few steps I was able to have NLP integrated into my bot. Choosing the Language understanding template the bot will automatically be set up and communicating with LUIS, which was eager to start predicting some utterances intents.

The code from Listing A.1, is a snippet of the MessageController class, this code was automatically created with the template, it is responsible for handling the arrival of user's messages. Every time that the bot receives a POST message, the bot retrieves the activity from the POST body, as represented in section 3.3.2.1, and checks for the activity type. For this bot we are only interested in the 'message' type, which represents that there has been a communication between the user and the bot, but we can still separate it between two 'message' types, if there is a parameter value in the message, it means it that the message is the response data from a card attachment or just a normal Text message.

In the card case, I decided to use Adaptive Cards [76]. The Adaptive Card format is a simple yet powerful declarative layout format that provides a lot of flexibility, allowing for visually rich cards and adds a more rich interaction for the user, even though these cards are simple JSON formatted messages. Adaptive Cards are the dominating standard for any Card-like UI in any of Microsoft's product, that is why the objective was to choose the visual option that could work with the most number of channels, unfortunately it made impossible for this chatbot to run on Facebook, since it one of the channels that are not able to render this type of cards [77].

Cards can be used for a variety of things, they can be just informative but they can also be used to request user input, this means that that will be sent back to the bot, in the form of value in the message activity. To submit card value to the bot, the card must include the submit button action, this action gathers all fields, merges with the optional data field, and sends the event to the bot. Since it is up to the developer to determine how this data is processed, to be able to handle the data submitted from each of these cards, and knowing how to distinguish between them, every submit action must include the parameter "type", that will define which card the information came from, this way I can define different data processing techniques for each card type, represented by c_type in Listing A.1.

Then there is the normal message type, the text message, where the bot will create a new LUIS dialog, the "BasicLuisDialog", which will be responsible for sending the request to LUIS, to predict the message intent, as well as, handle the intent prediction response that is sent back from LUIS.
4.6.2 LUIS Integration

The BasicLuisDialog class, from Listing A.2, is responsible for the connection with LUIS. For the bot to be able to connect with LUIS, you provide it with your LUIS application ID and the corresponding subscription key, the subscription key is the identifier of the developers Microsoft account and provides access to the API and the application ID is the identifier of the specific application. These are represented by the "LuisAppId" and "LuisAppKey" in Listing A.2, which should be stored safely in your application settings.

When the bot sends the activity message to the BasicLuisDialog class, this will forward this message to your LUIS application endpoint, an HTTP endpoint utilizing a REST API. The communication is done using JSON formatted data messages. A prediction of an utterance is simply made by sending a GET request to the application with the utterance attached and the subscription key and application ID must be attached as headers. LUIS will perform the predictions it was trained to, section 4.1, and returns the prediction answer, Listing A.3 represents a possible reply from LUIS.

LUIS response returns the text that the user sent, along with its prediction, where the 'topScoringIntent' will define the highest scored intent prediction, this dialog automatically decides what function to run next, depending on the defined 'topScoringIntent', the function will route the result directly to the function defines with [LuisIntent( topScoringIntent )], in the example's case, since the top score was for the 'Saldo' intent, it would be routed to the Saldointent function.

Knowing how this "intent routing" works, I had to create one function, and the corresponding route, for each intent that was created in section 4.1. When each intent as a corresponding function, all there is left is to define the way the LUIS result is treated, and how to treat each entity, depending on what they are for. For this project, we know, at least, that each intent should correspond to one of the BankOnBox operations.

4.6.3 Bing Spell Check API

The idea for using bing spell check was to handle some of the linguistic challenges that are faced while processing natural language, these challenges were presented in 3.6. But from implementing the API through LUIS the only challenge that the spell checker was able to do is actually just the spelling errors, and there were still some problems with this implementation to be referenced in this section. The objective was to implement a middle layer between the bot and LUIS, to send the misspelled sentences and provide LUIS with the correct utterance, so that LUIS would not notice that the user made a mistake, instead it processes corrected utterance assuming that it is the same utterance sent by the user, by having the spelling corrected LUIS will not need to be trained for different type of spelling errors.

To implement the spell check, create a new spell check cognitive service in Azure [78], in order to acquire the API service key. With this key, the code is shown in Listing A.2 was updated to include a different configuration to the LuisModelAttribute.
public static LuisModelAttribute LUIS_MODEL = new LuisModelAttribute {
    ConfigurationManager.AppSettings["LuisAppId"],
    ConfigurationManager.AppSettings["LuisAPIKey"],
    domain: ConfigurationManager.AppSettings["LuisAPIHostName"],
    }
    BingSpellCheckSubscriptionKey = ConfigurationManager.AppSettings["BingSpellCheckSubscriptionKey"],
    SpellCheck = true
};

Listing 4.13: Snippet from BasicLuisDialog.cs, initially created by the SDK, which a few changes

Listing 4.13 shows the LuisModelAttribute being updated to add the additional variables "BingSpellCheckSubscriptionKey", in which you provide LUIS the Bing Spell Check service key, that was mentioned previously, and for letting the LUIS application know that the utterance that should be reviewed is being translated, or corrected, and it should use that translated utterance to perform its prediction. This configuration will add the new parameters to to LUIS endpoint API URL, so that when a new utterance is sent, it will first go through the spell check, if Bing Spell Check API detects a misspelling, LUIS will receive the translated query, that will be used for the prediction, in response LUIS sends a different message, as can be seen in Listing 4.14, where the query field contains the original query, and the alteredQuery field contains the corrected query sent to LUIS, the word "tranferencia" was corrected to "transferencia".

This looked perfect but there are some occasions this has an impact on entity extraction, in my particular case, there were some strange cases there was the case with the name "ruben" that is part of a phrase list feature, mentioned in section 4.1, so I had expected that Microsoft would somehow exclude such words from the translation process since they were explicitly specified by the LUIS application owner. Unfortunately, at the time of writing, they do not pair the translation process with the phrase list features, and for some reason, some names were being corrected, in this case, "ruben" was being corrected to "rubem", there was also the case where if the user decided to some nickname or uncommon name, like the example, "jammy", the spell check would correct it to "jam my".

This problem might be caused because no language is being defined in this spell checker configuration, it is automatically detecting the language and making the translations not based on LUIS defined language.
4.6.4 BOB API

With LUIS being able to predict the necessary intents, and the bot already configured to call a different function depending on the intent, we can start implementing BOB API operations calls.

The API Calls are made via a REST API, with JSON formatted messages, i.e. making HTTP requests to the API Endpoint, where each request has to be made to a specific API Endpoint URL with the according path that corresponds to the operation o need to perform, as well as some requested parameters, depending on the request, for example the balance and movements request, are just HTTP GET requests and the only parameters needed is the desired account number for the operation, that is sent along with the request URL, and, on the other hand, a payment request is an HTTP POST request, where you need to send the payment information in a specific format, along with the HTTP request, this format was determined in the API itself. In return, all these requests return the corresponding data for the requested information, you can find an example of both types of request in Listing A.5 & A.6.

The user needs to prove that it is authorized to perform such action, this is done through the use of the BankOnBox OpenId Access Token, 4.2.1, this token should be sent in the HTTP request authorization header, without it the user is not allowed to perform any actions in the BOB API. Access Tokens have an expiration time, when this happens, the request to the API will return a message saying that the user is "Unauthorized", which means that the user needs to re-authenticate, or in this case, the Refresh Token can be used to re-authenticate the user without the user even noticing. To do this, the function in Listing 4.15 is called, which utilized the OpenID Connect configuration that was previously shown in Listing 4.1, to renewal the access token. If this function returns an error, the bot will assume that it was a communication error, as can be seen in Request Flows Figures, e.g Figure 4.10, because refresh token should always be valid. Otherwise, the result will be a new access token, as well as a new refresh token.

```csharp
var refreshed = await client.RefreshTokenAsync(refreshToken);
```

Listing 4.15: OpenID Connect, Refresh Token

4.6.5 User Authentication

In order to get the Access Token, the user must authenticate in the BOB's API, this authentication is done through OpenId, section 4.2.1, and is demonstrated in 4.9.

While the application was being developed locally, the user could freely navigate the Chabot, if any requests were made, the chatbot would automatically open a pop up tab on the user's browser, whenever it needed the user to log in, i.e, the log in was made, only, whenever the chatbot needed the Access Token, to perform a banking bob API request. But this option was not viable in a hosted environment, as it was mentioned in 4.2.1, the chatbot was not able to freely redirect the user to the login page. Having that in mind, and the fact that the chatbot needs the access Token to perform any of the chatbot operations, it was decided that the user should be required to log in at the very start, so, as long as the user tries to communicate with the bot while it is not authenticated, the log in prompt, marker 1.1 in 4.9, the log in prompt will keep showing up.
Another problem rose up during hosted development, at first, the user was automatically prompt as soon as it entered the chat window room, the conversation, but, when hosted, the bot behaved differently. When you enter the conversation, the user is given and userID, but as soon as the user sends a message, it would have another one, the one that was defined through the Web Chat JavaScript, from A.4. Because of that, the user now has to start the conversation, marker 0.1 in 4.9, in order to get prompt for login, marker 1.2.

By clicking on the BOB Login option, in marker 1.1, the state data is created, Listing 4.2, this data is saved in the state collection in the database, marker DB.a.1, as described in 4.3.3, meanwhile the user was redirected to the Login page, marker 2.a, if everything goes well with the log in, the state data will be retrieved from the state collection for confirmation, where if the confirmation is successful, the bot will receive the user’s Access Token. Having the Access Token the user can now start making requests to the application.

In case the user decides to use its Facebook Account, it will still have to log in in their Bob’s Account. The first time the user logs in into their Facebook’s account, the “facebookid” will not be registered in any of the Accounts in the Account Collection, mark Db.b.1, this will cause the bot to prompt the user to link their accounts, mark 3.b, by re-logging in their BoB’s account, which will cause the “facebookid” to be added to the corresponding BoB Account Document, mark DB.b.3. From here on out the user should be able to simply log in to their Facebook account, and retrieve the Access Token from the corresponding file that is found in mark DB.b.1, without having to log in with their BOB’s Account ever again.
Figure 4.9: Dialog Flow, Login
4.6.6 Chatbot Operations

In this section it will be explained the thought behind the design of the chatbot, its interactions with the user, changes that appeared during development, and showing off the capabilities of the chatbot. For each Operation it was created a Conversation Flow, these will provide an idea of all the different elements working together and exactly when are they being used, the previous section already used one example of this Flows, although, there was actually no real conversation.

4.6.6.1 Balance Request

In the same way, we started, by defining the Balance Intent "Saldo" in LUIS, 4.1.1, let focus on the Balance Request first, which honestly is the simpler one. For this section please refer to Figure 4.10

As was said in section 4.1, Saldo has only 2 possible input options. First, the balance without any specific account request, marked as 0.a, at first when you did not specify any account, the bot would assume that you wanted to see the defined main account, "mainAcc", that is retrieved from DB.0, but it might have been too confusing for the user, he might never remember what account it predefined, so it was decided that the user should have the option to see what it wants to see. So instead of using the main account, the bot uses the user's Account List, "ContasBancarias", also from the botcollection, marked db.0. Retrieving the values from that list, a Card with the accounts will be presented, mark 1.3, from which the user can pick and click on one of the accounts, to request the balance of, or he can opt to show the balance from all account, by clicking "Mostrar Todos". The second option is the user can say from which account it wants to see the balance, this will make the bot search the Accounts List, in search of a bank account that contains or matches the name given, if the name is not found, the bot will then show the full list of accounts, like usual, mark 1.3, otherwise it will execute the API call shown in Listing A.6, which needs the access token, that is retrieved at the beginning of any conversation, also from DB.0, and the account that was announced by the user. Picking an account from the list will result in the same action, but basically with one less step. The difference here is, if the user picks to show the balance from all accounts, a different API request is made, this request asks for the list of accounts, but with a special parameter in the URL request, "acctlist?withBalance=true", which provides us with the list from all the accounts, with their corresponding balances, instead of having to make one call for each of the user’s account, saving time and resources.
Figure 4.10: Dialog Flow, Saldo
4.6.6.2 Movements Request

The Account Movements is represented by a list of banking transactions, as one is used to seeing in our everyday banking systems, the minimum information provided for each movement is the starting and ending transaction date, its description, the amount and the nature of the movements (credit or debit). Knowing that the BOB API is limited to showing only the last 20 movements, that is all I had to work with, never the less, the user should have the opportunity to see only the movements it wants. Unfortunately, BOB API does not accept any other parameters for the movements request, there are no movements filtering, meaning the bot has to do it itself. To do this the bot must request the API for the list of transactions, analyze each transaction one by one, checking if the transaction information’s meet the filtering options of the requirement. These filtering options were also defined according to the available transfer information, which are the transfers’ date, start and end date, and the movements’ nature, credit or debit. In order to be able to filter these options, the LUIS model had to be trained accordingly, by having, and being able to identify entities for each of the previously listed features, section 4.1.3.

For this section please refer to Figure 4.11, the Movements Dialog flow, mark 0.a, the user can make a request without any of these filters, as explained the API request is still the same, the bot retrieves the user account data at the beginning of the conversation, mark 1.2, having retrieved the necessary main account, “mainAcc”, because no account was defined in the request, and the access token, the bot can initiate the API request, mark 2.1, which will return, if successful the full list of the user movements, or at least 20 of them, this list is presented a Card with a similar look that you would expect from other banking systems, or the ATM, which shows the result information organized by date, description and finally amount, either credit or debit, as seen in mark 3.1.

On the opposite side, mark 0.b, the user may also make a more specific request, if the date entity “Data” was found the data will have to be filtered by date, the hardest part for me was not being able to recognize the date in the utterance, but what exactly to do with each entity, to be able to achieve the desired effect. To be able to compare dates, the DateTime data type has to be used to work with date and times, in C#. The DateTime class in C# provides properties and methods to format dates in different DateTime formats and also direct comparison. The first problem is knowing what do do with the different elements that can be a date, for example, if the date is “today” or “yesterday”, DateTime is used to retrieve the current date, for today, and the current day minus 1 day, for yesterday. Then you have to correctly format the date to be able to parse it as a DateTime data type so that you can compare if one date is greater or lesser than the other. Finally, you have to understand the range for each date request, because the user is not obliged to provide a full date, meaning a year, month and day, it can freely choose, i.e, the user could also pick to filter only by the year 2012, where it every transfer from 2012 is presented For the credit and debit, the application just has to check if the transaction amount signal was either positive or negative. The application uses these filters to either add or not, the transaction to the Movements Card, mark 3.1, in Figure 4.11.
Figure 4.11: Dialog Flow, Consultas
4.6.6.3 Payment Request

Last but not least, the Payment request is the one that actually has required parameters that are defined by the user. The payment, which is actually more a transference, requires the user to tell the bot three parameters, that can be seen in A.5 the payment sets 3 object variables, the CreditorName, that is actually just a description for the transfer, the creditor account, the IBAN number to whom you want to make a payment to, and obviously, the amount. As explained in section 4.1.2, the model was trained to recognize the IBAN and the amount value. If any of this entity is found, the bot will use them to fill the Payment Card the input fields for the user, as shown in mark 2.b in Figure 4.12. The IBAN is always in the same pattern, so since it is a regex entity, it directly matches the entity if the pattern is matched, so it is guaranteed to work. The amount was harder to implement, since an amount value can be either a number or written in full, using the number prebuilt entity, had some clashes with some words that were not meant to be an amount, like in the utterance “fazer uma transferencia”, in English “make a transfer”, the word “uma” is defined as a number, and there is no way to not label it that way, since prebuilt entities are always gonna be labeled if found, so I had to create a line of code just to ignore the word “uma”, other than that, it is really useful for identifying numbers written in full, and since it evens provides the integer number along with the prediction, the bot is able to also fill this field for the user.

Seeing that no one wants to memorize IBAN numbers, nowadays most of the people might not even remember their own phone number, and that is only 9 digits, the IBAN has 25 but since “PT50” is the beginning for any Portuguese IBAN, 21 digits would be even worst. That is why the Contact List was implemented, the contact list keeps you saved IBAN numbers, which are associated with a name chosen by you, which will be explained later. The payment recipient entity, “destinatario”, was implemented in LUIS, so that the user could say the name of the contact, instead of the full IBAN number.

So if any of these entities are used, the bot will fill in the corresponding fields, mark 2.b, in Figure 4.12, and the users just need to insert the description. The user can check the account it is using in “Utilizando conta:”, but users are also given the option to change the account that their currently using, by clicking on “Mudar de Conta”, mark 1.b, in Figure 4.13, it will open a new section of the Payment Card, that shows the users accounts and the corresponding balance, from which the user can pick the desired account to use. The balance was decided to be shown so that the user can be sure of its monetary capabilities when making a new payment.

To prevent the user from making a meaningless request, a request that the chatbot is sure that will come back as an error, the user checks if the input fields were correctly filled, i.e. the description and the amount cannot be empty, the IBAN cannot be empty, less than 25 characters long, it should start with 2 words and that the rest is all numbers. Since the input field for the amount, "Montante", is a number input I did not have to confirm that it was not a word.

If the fields are filled correctly, the bot initiates the Payment Request, mark 2.1, in Figure 4.13, with the parameters defined by the user, if the API request is successful, an SMS confirmation message will be sent to the user, and the bot receives a payment id, which is stored in the Confirmation Card, mark 3.1, in Figure 4.13, this card is filled with the payment parameters input by the user and the confirmation
code, from the SMS, should be entered to confirm the payment.

For development purposes there was not really any SMS code being sent, the code was always the same, even so, we had to think about what could happen in a real environment, so there is is a possibility that the SMS code does not reach the user, that is why the option to resend the code is present, to simulate this all this does is remake the same payment as before, which in theory would resend the SMS.

To confirm the payment the user has to input the 4 digit code, which is also checked if it was filled correctly, can be empty and must be exactly 4 digits, else it keeps warning the user that the field is not correct. But before confirming the request the user should make sure that the payment request is correct.

The bot retrieves the payment id that was saved in the Confirmation Card, mark 0.1, in Figure 4.13, and makes the necessary API request. From experimenting, if the user enters the wrong SMS code once, the payment will return that success was false and the payment is automatically canceled, meaning that even if the user retries with the right code, it will not work, and the user will have to start the whole payment again, the bot will warn you about this, mark 2.b in Figure 4.14

But, if the payment is, in fact, a success, the bot checks if the user defined the recipient entity “destinatario” in the request, but the bot was not successful at founding the recipient on the Contact List, the user will be prompt to add the recipient to your contact list, with the name that you entered as the entity and the IBAN you used for the payment, as in mark 2.a.a, in Figure 4.13. Else, the bot checks if the input IBAN is already registered in your Contact List, which if it does not, the bot prompts the user to input a name to give to the IBAN, in order to create a new contact. The user is never obliged to actually respond to a card, but if the user really wants to add a new contact, it just has to fill the name correctly, with a name that is not already on your list. If a new Contact is created the bot will update the user’s account data information, in both DB.2 & DB.3, in Figure 4.13.
Figure 4.12: Dialog Flow, Pagamento part1
Figure 4.13: Dialog Flow, Pagamento part2
Figure 4.14: Dialog Flow, Pagamento part3
Chapter 5

Results

As a result, a fully functioning chatbot was created, all the necessary operations were implemented, for each of these operations LUIS corresponding intents were trained. For this training it was applied a “learning by doing” approach, meaning that there was no actual data set provided or that could be found to inject banking operation utterances for LUIS training, it was all done by hand, different ways to talk and to make the different requests were used. This training would have benefited from a bigger set data that could have been collected from banks, that would have the utterance that customers’ have been using to ask for their banking operations, so from this training LUIS ended up with the results presented in Tables 5.1 & 5.2

<table>
<thead>
<tr>
<th>Intents</th>
<th>Labeled Utterances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saldo (Balance)</td>
<td>75</td>
</tr>
<tr>
<td>Movimentos (Movements)</td>
<td>215</td>
</tr>
<tr>
<td>Transfêrencias (Payments/Transfer)</td>
<td>111</td>
</tr>
<tr>
<td>None</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 5.1: LUIS Intents Training Results

<table>
<thead>
<tr>
<th>Intents</th>
<th>Entity Type</th>
<th>Labeled Utterances</th>
</tr>
</thead>
<tbody>
<tr>
<td>saldo</td>
<td>Simple</td>
<td>73</td>
</tr>
<tr>
<td>contaBancaria (Bank account)</td>
<td>Simple</td>
<td>55</td>
</tr>
<tr>
<td>consulta (consult)</td>
<td>Simple</td>
<td>156</td>
</tr>
<tr>
<td>Data (Date)</td>
<td>Composite</td>
<td>141</td>
</tr>
<tr>
<td>ano (year)</td>
<td>Simple</td>
<td>91</td>
</tr>
<tr>
<td>mes (month)</td>
<td>Simple</td>
<td>111</td>
</tr>
<tr>
<td>dia (day)</td>
<td>Simple</td>
<td>100</td>
</tr>
<tr>
<td>credito (credit)</td>
<td>Simple</td>
<td>20</td>
</tr>
<tr>
<td>debito (debit)</td>
<td>Simple</td>
<td>20</td>
</tr>
<tr>
<td>transferir (transfer)</td>
<td>Simple</td>
<td>115</td>
</tr>
<tr>
<td>dinheiro (money)</td>
<td>Simple</td>
<td>58</td>
</tr>
<tr>
<td>IBAN</td>
<td>Regex</td>
<td>N/A</td>
</tr>
<tr>
<td>destinatario (recipient)</td>
<td>Simple</td>
<td>66</td>
</tr>
<tr>
<td>number</td>
<td>Prebuilt</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 5.2: LUIS Entities Training Results
In order to improve and get some feedback about the bot performance, it was proposed by the project coordinator that the application should be tested by some users that were more experienced on the field, either chatbot or banking experience, since the coordinator had been accompanying the project, he wanted to see what someone with chatbot experience would think about the Chatbot’s performance as well as knowing what could be done differently or improved.

Unfortunately, for timing issues, we could only reach out and collect feedback from a short number of users, it was only tested by five real experienced users. Despite the low number of users, everything user interaction with the chatbot was used to further train the chatbot NLP capacities and also get some make some adjustments to the solution, according to the users’ feedback.

In order to do this test, the bot was deployed and hosted on a Virtual Machine running in Microsoft Azure. As was explained in section 4.4, these users had access to the deployed Chatbot through the web site, “https://lasttestlinkbot.azurewebsites.net/”, and they were provided with the BankOnBox Login credentials. This login credentials also created a bit of a problem, BOB API is not open to the public, a new account can not be created by the user, so they were forced to use the given credentials.

The users were reached by email and were only given a basic explanation of what the bot is and does, all it was said was that the chatbot was supposed to perform banking operations and they should follow the chatbot instructions, in order to freely experiment the bot capabilities. But this test was not observed or controlled at all, many of the users did not try all the different dialogs and different requests they could make, so I will not be actually presenting statistics from the test, because there was not much to go with but also because the tests were so different that it did not really provide any relevant information.

The results were studied and the user provided feedback, which will be presented next with some comments on them.

User Testing Feedback

- Chatbot Functionality:
  - First they discovered spelling errors in our chatbot responses, that were quickly corrected.
  - The Movements Operation should, in the same way as the Balance, ask what account would the user like to see the movements from, not because it is necessary but to keep the coherence between all operations, because both balance and payments allow the user to choose the account before performing the operation, so the user will expect to be able to do the same for all operations.
  - Some of the users expected that there would be a payment by reference and not only with IBAN, because that what we are used to when making a Payment, unfortunately, this option cannot be implemented, because this is the only option that was implemented in BOB API.
  - It was purposed that the authentication methods could be done inside the web chat, and not open another tab. This option was considered, but the problem with this was that there was no way to emulate a browser window, all that could be done would be to ask the user to fill
cards fields that would represent the username and password inputs, but unfortunately, there was no masking option for card inputs so that possibility was discarded.

- The Web Page should adapt to the user platform, or at least to not stretch as wide, limit the chat windows. This would provide standardization for the appearance of our application, so we can make sure that the application works and appears in the same way for all users.

- Utilization of “quick replies”, when the user logs in, the welcome message should list possible quick reply actions, balance, movements, and payments. With this, the user can quickly click on the operation it wants, instead of having to always write, or talk. It also lets the user know what operations the bot is able to do.

- There should be more options to skip the need for cards, like to show all the balances from all the accounts, instead of having to click the card. This would be great for regular users, so that they can use the application quicker, for example in the payment option, if you provide the IBAN and amount, the payment could automatically start, instead of showing the filled payment card, the user would not have to fill in the payment card, it would automatically go to the payment confirmation step.

- Some professionals did not like the way that the bot talks so casually and think that for this kind of application the conversation should more formal. I am not very accustomed to being formal, so that is absolutely something that I should improve on.

- Some users had an Exception, an error, been thrown directly in the chat, that is inadmissible and has been corrected immediately. More testing should be done, in order to ensure that no exceptions or errors are not being handled, and make sure they are handled properly.

- The Balance card with all the accounts list was not clear that the accounts could be clicked in order to check their balance, this alteration can be seen from Figure 4.10 to Figure 3.2, where we provided additional information for the user to know to click one of the accounts options to select it.

- Users were confused because the default account was select for them, without any previous questioning. Since users were using the same BOB account that I had used for all the development, the default account was already select by me, not by the current user, but it should have been implemented a way to check the available account and the details for each of them, because the users wanted to know their banking information, and contacts lists.

- While testing the payment method users asked the bot for the IBANs or something to help them, because they did not know what number they could use, or wanted to try inner banking transfers. The bot should provide an explanation when the user asks for elements of the current operation, or the contact list should be listed, some users did not even make a payment request for this reason, they quit without trying a random value.
• Natural Language Processing:
  – User’s experienced some problems when asking for a specific bank account, mainly in the movement’s operation, meaning, that LUIS failed to identify the bank account, this was caused by a difference from my speech pattern and theirs. One problem was that some were trying to write a very detailed and long message, it showed that LUIS was getting confused with the full account names. The other would be that the Patterns that were trained in LUIS were not matching the speech pattern of the users. The users have some background on the chatbot area and were using a different vocabulary that LUIS was not trained for, since most of the training was actually done manually. More users would help LUIS understand different speech mannerisms.
  – Adding more basic intents like a goodbye, for when the users say it is leaving, for when the user asks who the bot is, or what its name, how old are you, simple entities that the users are expected to be basic and should have been there. There should have been a better understanding of the users’ needs, and not only what the bot needed in order to operate.
  – There are still many other types of date request to consider, for example, “two weeks ago”, or “from five months”, many of these date patterns were not considered during implementation, and new ways to ask for a date were sure to appear with further testing.
Chapter 6

Conclusion

6.1 Conclusions

Many forms and structures are experiencing changes which Machine Learning, Artificial Intelligence, and Chatbots are bringing. The ways we process things, issues and life are going to change.

Chatbots have been used in the form of online assistants in corporate web sites. The youth is mobile-oriented and appreciate instant messaging in everything, they want quick and efficient answers. Chatbots are always ready to respond immediately, they help businesses reduce the customer servants workload in answering customers’ questions, or commands, and at the same time serve the customers in the best possible way, always.

This project presents a solution to developing a chatbot for financial institutions. The concept of Open Banking was studied in this thesis, and with the changes to the PSD2 legislation, banks have to expose their interfaces to interested third parties, that why many banking third-parties applications are expected to start appearing.

Banks are looking for entities that can provide solutions to these changes, and companies that other companies that offer Internet Banking products are doing the same, such as Link Consulting, with its Internet Banking product, BankOnBox.

This project allows financial institutions that take advantage of BankOnBox solutions to be one step ahead of the competition, by providing users a new way to interact with their services, which are no longer stuck with using the institutions’ applications. The chatbot is able to perform the banking actions on behalf of the user.

It was also studied what chatbots are, as well as the needed components to develop one, which leads to, developing the chatbot by taking advantage of one of the available Chabot platforms, to aid the chatbot creation process as well and integrating all the necessary NLP and speech recognition capabilities. Different solutions were studied and considered, the state of the art, and finally, Microsoft Bot Framework is one of the current top platforms in the market, was the platform elected.

This project took advantage of the easy integration of cognitive services, through the use of this platform, adding natural language processing (NLP), using LUIS linguistic capabilities, as well as integrating
speech recognition to the chatbot, using Bing Speech API. Both these tools were also chosen for their compatibility with the required language, Portuguese.

With NLP it is possible to make the interaction with the customer more conversational. The NLP tool used, LUIS, allowed the chatbot to understand user commands, making use of intents and entities, in order to understand users requests and to provide the bot with the context banking operation. The chatbot communicates with BankOnBox’s API, BOB API, in order to perform these banking operations on behalf of the user. The applications also implemented Bing Speech API services, which enable the bot to recognize speech, providing the user with a different way to communicate with the chatbot.

The objective of developing a chatbot application was successfully achieved, the bot was made accessible to the public, a web page was created solely to interact with the chatbot, but the chatbot could also be embedded in a banking institution web page. It can understand users’ requests for banking operations, like proving the user’s accounts balance, movements transactions, and payments, and could be trained to perform much more in the future. This version of the bot produced the basic functionality, but it can be extended based on users’ needs and feedbacks.

6.2 Future Work

Even though the system was implemented and is able to perform the intent banking operations, there are still many improvements that can be made:

- A bigger test should be done, a chatbot success is based on the user experience, and perform a controlled evaluation of the chatbot capabilities.
- LUIS could be trained with many more utterances, this would benefit by having users utilize the chatbot in order to better train the language processing model.
- Testing of the Speech capabilities that were implemented in the bot, which was not given its proper value, this should be tested for acceptable noise levels, user accents, and overall speech recognition.
- Adjustments should be made considering the user’s feedback in section 5.
- Bing Spell Check capabilities were also not properly tested, demonstration of the API capabilities should have been shown.
- The most challenging aspect of this project was understanding the difficulties of NLP. This was underestimated, and as a result, some linguistic challenges were not implemented, like the homophones, or predicting what the user wanted to say, given the conversation context, word abbreviations. These challenges should have been more investigated in order to create a chatbot that could fully understand Portuguese. Although the chatbot is able to operate, it is not fully capable to handle every user input or to correctly identifying the intent behind it.
- Implementing the chatbot in different messaging channels, like Skype, Cortana, Microsoft Teams, Cortana, Slack, Telegram or Facebook Messenger, which was actually tried, but Messenger did not support the choice of cards that were implemented, the Adaptive Cards, so there might be some features that have to change depending of the channel being used.
Bibliography


Appendix A

Listings Appendix

```
[BotAuthentication]
public class MessagesController : ApiController{
    [ResponseType(typeof(void))]
    public virtual async Task<HttpResponseMessage> Post([FromBody] Activity activity)
    {
        // ....
        // check if activity is of type message
        if (activity.GetActivityType() == ActivityTypes.Message)
        {
            if (activity.Value == null)
            {
                // checks if user is logged in
                if (String.IsNullOrEmpty(accessToken))
                {
                    // if not, requests login
                    await operations.ReqLogin(activity);
                    return new HttpResponseMessage(System.Net.HttpStatusCode.Accepted);
                }
                // sends activity to LUIS
                await Conversation.SendAsync(activity, () => new BasicLuisDialog());
            }
            else{
                dynamic cardData = JObject.Parse(activity.Value.ToString());
                string c_type = cardData.type;
                // ....
            }
        }
        else{
            HandleSystemMessage(activity); }
        return new HttpResponseMessage(System.Net.HttpStatusCode.Accepted);
    }
```

Listing A.1: MessageController.cs code snippet, automatically created by the SDK
public class BasicLuisDialog : LuisDialog<object>
{

public BasicLuisDialog()
    : base(new LuisService(new LuisModelAttribute(
        ConfigurationManager.AppSettings["LuisAppId"],
        ConfigurationManager.AppSettings["LuisAPIKey"],
        domain: ConfigurationManager.AppSettings["LuisAPIHostName"]
    ))

    {

        [LuisIntent(“Saldo”)]

        public async Task SaldoIntent(IDialogContext context, LuisResult result)
        {
            await context.PostAsync($"You have reached {result.Intents[0].Intent}. You currently have 420");

            context.Wait(MessageReceived);
        }

    }

Listing A.2: Snippet from BasicLuisDialog.cs, initially created by the SDK, which a few changes

Listing A.3: LUIS response
Listing A.4: Chatbot Webpage, HTML File.
PaymentModel payment = new PaymentModel();
payment.Amount = valor;
payment.CreditorName = _cname;
payment.CreditorAccount = _iban;

// account number example
var accountnumber = "8e69750b–0cd9–4166–a8a3–cb01525a7f0a";

var json = JsonConvert.SerializeObject(payment);
string reqPath = "";
westeurope.cloudapp.azure.com/ais/initpay/" + accountnumber) { Content = new StringContent(j
son, Encoding.UTF8, "application/json") };
accessToken);
reqMsg.Headers.Add("Ocp-Apim-Trace", "false");
reqMsg.Headers.Add("Cache-Control", "no-cache");
reqMsg.Headers.Add("Ocp-Apim-Subscription-Key", SUBSCRIPTIONKEY);

HttpResponseMessage content = client.SendAsync(reqMsg).Result;
string JSON = await content.Content.ReadAsStringAsync();

Listing A.5: BOB API, HTTP POST example, Payment Request

var clientx = new HttpClient();
clientx.DefaultRequestHeaders.Clear();
clientx.DefaultRequestHeaders.Add("Accept", "application/json");

// account number example
var accountnumber = "8e69750b–0cd9–4166–a8a3–cb01525a7f0a";
westeurope.cloudapp.azure.com/ais/acctbal/" + accountnumber) { Content = new StringContent(j
son, Encoding.UTF8, "application/json") };
accessToken);
reqMsgx.Headers.Add("Ocp-Apim-Trace", "false");
reqMsgx.Headers.Add("Cache-Control", "no-cache");
reqMsgx.Headers.Add("Ocp-Apim-Subscription-Key", SUBSCRIPTIONKEY);

await context.PostAsync("A efectuar pedido de " + result.TopScoringIntent.Intent);
string JSON = await content.Content.ReadAsStringAsync();

Listing A.6: BOB API, HTTP GET example, Account Balance Request
Chatbot Banking Request Examples

SALDO
Ver o saldo...
Da conta < contaBancaria >

CONSULTAS
Consultar os movimentos...
De hoje
De ontem
De anteontem
De <dia>
De <mes>
De <dia>/<mes>/<ano>
De <dia> <mes>
De <dia> <mes> <ano>
De creditos (Money In)
De debitos (Money Out)

PAGAMENTOS / TRANSFERENCIAS
Fazer uma transferencia...
Para <IBAN>
Para <destinatario> (nome)
De <montante>
De <montante> para <IBAN>
De <montante> para <destinatario>
Para <IBAN> de <montante>