INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Wegho Chatbot

MIGUEL JOÃO COELHO DA SILVA
Outubro de 2018

POLITÉCNICO
DO PORTO

Wegho Chatbot

Miguel João Coelho da Silva

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: António José Rocha de Oliveira

Júri:

Presidente:

Vogais:

Porto, outubro de 2018

Resumo

Esta dissertação foi escrita para acompanhar o projeto de construção de um Chatbot para o Marketplace Wegho, de a modo melhorar o seu serviço de Apoio ao Cliente. Este Chatbot foi desenvolvido utilizando o Azure Bot Service da Microsoft.

A Wegho é um Marketplace que oferece tipos diferentes de serviços como limpeza doméstica e reparações, e que funciona através de um website e de aplicações móveis, disponíveis para iOS e Android. Estes serviços são disponibilizados através de trabalhadores profissionais e qualificados. Através do Marketplace um cliente pode marcar um ou mais serviços para uma data aceite entre as duas entidades, por um preço competitivo.

O Chatbot desenvolvido tem como propósito principal resolver simples problemas que um cliente possa ter, tal como responder a questões frequentes. Também é capaz de gerir serviços, seja a criar um novo ou a alterar ou cancelar os que já estiverem previamente marcados.

Este serviço funciona através de conversas naturais com o cliente, como se este estivesse a falar com outra pessoa, percebendo o que lhe é solicitado, qual o objetivo desejado com cada mensagem, e qual a informação enviada. Nos casos em que o Chatbot não é capaz de "compreender" o que o utilizador necessita, é capaz de entender que é o caso e então redirecioná-lo para um agente humano do serviço de Apoio ao Cliente.

Este serviço foi avaliado através de um período de testes, tendo utilizadores de teste a falar com o Chatbot com objetivos específicos, e gravando as transcrições destas conversas. No fim de cada conversa o utilizador teve também a possibilidade de dar feedback para que possa, posteriormente, ser analisado. Com estas transcrições outras métricas foram retiradas para avaliação, tais como a qualidade dos pares de mensagens entre o utilizador e o Chatbot, a quantidade destes, se as respostas são suficientemente diretas ou demasiado ambíguas, entre outras.

Palavras-chave: Chatbot, Inteligência Artificial, Linguagem Natural, Marketplace

Abstract

This dissertation was written to support the project of the construction of a Chatbot service for the Wegho Marketplace in order to improve the Customer Support service. This Chatbot was built using Microsoft's Azure Bot Service.

Wegho is a Marketplace that offers different kinds of services such as domestic cleaning and home repairs, and works through a website and mobile apps, on iOS and Android. These services are provided using professional and qualified workers. Through the Marketplace a customer can schedule one or more service on a suitable date, for a competitive price.

The Chatbot developed has as its main purpose the solving of simple customer issues, such as answering frequently asked questions. It is also able to handle some service management, such as creating a new one, or modifying and cancelling previously existing ones.

This service works through natural conversations with the customer, understanding what is required of him, what the goal intended with each message sent to it is as well as the information sent. In cases where the Chatbot doesn't "understand" what the customer wants of it, it recognizes it and redirects him or her to a human agent within the Customer Support service.

This service was evaluated through a test period, having test users talk with the Chatbot with specific goals, and saving the transcripts of these conversations. At the end of each conversation the user also gives feedback so that it may later be analysed. From these transcripts another metrics to then be evaluated were taken, such as the quality of utterance pairs between the user and the Chatbot, the quantity of these, if the responses are direct or more ambiguous, among others.

Keywords: Chatbot, Artificial Intelligence, Natural Language, Marketplace

Index

Figures List

Tables List

Code Excerpts List

Acronyms and Symbols

List of Acronyms

LUIS	Language Understanding Intelligent Service
QnA	Questions and Answers
SDK	Software Development Kit
AI	Artificial Intelligence
FB	Facebook
CRM	Customer Relationship Manager
SMS	Short Message Service
ETA	Estimated Time of Arrival

1 Introduction

In this section an introduction will be made on the project elaborated for this dissertation. In it the context will be described, along with its problems and objectives, as well as the approach chosen for its solution. Lastly, the structure of this document will also be described.

1.1 Context

Wegho is a marketplace of domestic services, using properly validated suppliers of these kinds of services, and working through a website and mobile application. The Customer Support is a very important factor for these types of services, and even though a frequently asked questions zone already exists, much of these works end up being done by human assistance. This support service needs to be able to handle every client or potential client, helping him/her with any situation they might come across. This may correspond to complex situations that need a personalized support, or something a bit simpler like providing the hyperlink to a FAQ webpage.

1.2 Problem

The customer support side of a service provided by a company is a fundamental piece of its organization. At Wegho there is currently one person dedicated to that function, helping every customer or potential customer in whatever is needed, either scheduling a new service or rescheduling one that can no longer happen as previously set. Adding to this important function, this person must also answer every small or big question anyone may have about the Marketplace, how it operates, what is currently possible or impossible to do. Many a times, the only way to help the inquirer is simply to direct him or her to the frequently asked questions section of the website or application.

Having these small or simple issues to be handled by the human customer support service can end up being a hindrance on the work that needs to be done, since more time could be allocated to handling more complex issues, such as those that cannot be solved by a simple answer or direction.

1.3 Objectives

With this project it's intended that a Chatbot for Wegho is developed. This will have to be able, through conversational interactions, to respond to simple requests from clients such as answering questions about the functioning of the marketplace, or just making a simple management of services. This management will include the creation of services with attributes given by the client as well as the maintenance or cancelation of previously created services.

This Chatbot will also have the functionality of assisting a service supplier when this doesn't "check-in" to a specific service, 15 minutes after it was scheduled to begin, giving him several options on how to solve this situation.

Lastly, the Chatbot will have to be able to understand the user and the state of the conversation adequately so that it can direct it to a human agent of the customer support, every time it understands that it won't be able to single-handedly respond to the user's requests. The user may also directly request to be recommended.

The Chatbot will be available on several communication services for its use, such as Facebook Messenger, Skype, Twilio, and on a Webchat on the Wegho website and mobile app.

In summation the objectives to be fulfilled with this project are the following:

1. Creation of a new service;
2. Modification of an existing service;
3. Cancellation of an existing service;
4. Assisting supplier when no check-in has been made;
5. Understanding of the user;
6. Direction of user to customer support when it fails to understand his/ her intent;
7. Direction of user to customer support when directly requested;
8. Availability of Chatbot in multiple channels.

1.4 Approach

To develop this project a study of the existing frameworks and tools were made, analysing their characteristics to choose what the most appropriate were. These Chatbot creation tools will have to be able to connect to multiple channels, specially Facebook Messenger, Skype and Twilio. They will also need to have the ability to be developed for the Portuguese language, since the final product will be used by a Portuguese company.

A connection to Zendesk, a Customer Service Manager platform will also be made, creating tickets, issues to be solved, anytime that it is pertinent to do so. These will have to be created when some sort of human interaction will have to be added to the Chatbot.

After choosing the technology to work with, the developer will use tutorials to learn how to properly use it. Guides and articles will also be made use of so as to know what the best practices when developing a Chatbot will be. These practices will not only be best software development patterns, but also conversational solutions patterns.

1.5 Structure

After this introduction this dissertation will have a section of Value Analysis, in which a study of what value the solution made will have to the actors that will end up using it. To do so, the Key Elements of the New Concept Development Model by Peter Koen will be analysed (Koen, et al., 2014). A value proposition will also be made, and a CANVAS model will be designed.

The section that comes next is the State of the Art, where various Chatbots creation tools are studied and described. At the end of this section there is an area where these tools are compared to have a contrasting view between them.

Afterwards there will be the section of Analysis, where through a Use Cases diagram all the tasks that the solution will have to perform are properly described and assigned to their respective actors, noting where associations between these tasks exists.

The following section will be the Design, where the various technologies studied are looked at deeply and described on how they were specifically utilized. In this section the various workflows of the solution are also described, through the usage of activity diagrams. It's here also that some of the design alternatives that were studied are described and explained on why

the decisions made were chosen. And finally, the architecture of the final solution is presented in the form of a components diagram and properly described.

The next section will be the Development, which is where the whole development of the solution is described. This includes how every component was constructed, what the main problems that arose were, and if and how they were solved.

On the section of Experimentation and Evaluation it will be described the ways on how the solution constructed for this project will be evaluated, such as what kinds of tests will be done and how their results will be studied.

Lastly, the section of Conclusions will be where a final balance on the project is made, reviewing what the objectives were and if they were accomplished, and having also an analysis of the limitations found when developing this project and the potential future work to be done.

2 Value Analysis

In this section an analysis of the value of the project developed for this dissertation will have on the client's business will be made. One of the main goals of this analysis is to find out what is important to develop, being able to discard features that would not have any use for the client, therefore increasing the efficiency of the project.

2.1 5 Key Elements of NCD Model

The NCD (New Concept Development) Model (Figure 1) serves to provide a common language and definition to the key components of a project or product, since its inception, from the opportunity identification or idea generation, to its final concept definition. In it there are five key elements, distributed in a wheel to demonstrate that these ideas flow, circulate and iterate among themselves. The arrows pointing into the model represent the starting point for projects, and they leave the front-end process by entering the New Product Development (NPD) or Technology Stage Gate (TSG) process.

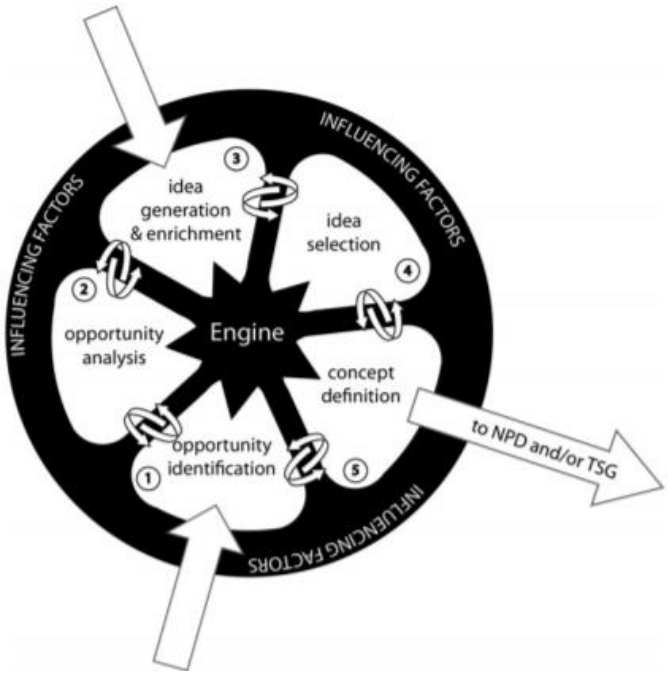

Figure 1 - New Concept Development Model Wheel

In this model the most important keywords are opportunity, idea and concept. An opportunity is a business or technology gap that exists between the current situation and an envisioned future. An idea is the most embryonic form of a new product or service, and a concept refers to a well-defined product or service with main features and customer benefits clearly identified.

2.1.1 Opportunity Identification

The Customer Support service available for the Wegho marketplace is provided solely by human agents, and a small and simple FAQ section in the webpage and mobile app. These agents are responsible for everything that may be asked of a customer, from helping create a new service, modifying or cancelling an existing one, or answering any questions they may have, no matter how simple or complex they might be. Many times, they must answer the same exact question to multiple clients, making the job unnecessarily repetitive, and making them waste time that could be directed to more complex and specific issues.

2.1.2 Opportunity Analysis

Since the Customer Support is handed exclusively by human agents, and it ends up having lots of repetition in the customer's issues to be solved, much of this could be solved by having the solutions accessible to the user in a much easier way. Some of these can be found at the frequently asked questions section of the website and mobile app, but most need some personalization towards the specific client, the issue, and its characteristics.

2.1.3 Idea Generation and Enrichment

To address the opportunity found, one solution is the creation of a system that automatically understands the issue to be solved by the Customer Support. This will need to have access to all the intricacies required to the solution of the specific problem, including the customer's attributes. With these, a solution to the customer's issue will need to be calculated, hopefully without the need for human interaction on the provider's side and be presented to the customer.

However, there will also have to be the need for the solution to be aware of the complexity of the problem at hand, knowing when the issue requires the assistance of a human, and therefore redirecting the customer to the Customer Support agent.

One way to potentially solve this issue could be to simply hire more agents to work on the Customer Support service, hence having more collective time to allocate all issues required. But this solution would only increase the cost of the whole service and would still mean that all customer's issues, big and small, would have to be handled by a human.

Another way to solve the issue at hand would be to develop a Chatbot, a natural language understanding software, that would understand the problem and solve it without the direct assistance of a human, unless required or directly requested.

2.1.4 Idea Selection

The idea selected was the Chatbot development. This was chosen because although there would be a cost to its development, this would be fixed and while one human has a limit to how many issues he or she can solve, the software would have no such limit. The Chatbot would be able to answer all the issues it was previously trained for, not being limited to one at a time.

This solution would also mean that the smaller and simpler issues a customer may have could be all solved by it, having no need to be handled by the human agents, clearing their time for more pressing issues.

2.1.5 Concept Definition

A Chatbot is an interface with which a person can interact with through natural language conversations, being able to both understand what is required of it and respond accordingly. This can be accessible through a popular messaging service such as Microsoft's Messenger or Skype, or directly in the website or mobile app in a webchat interface.

2.2 Value

2.2.1 Value for the Customer

The service designed with this project has its significance in that the customer will be able to have its issues solved in a simpler and more effective way. This service will also have an added functionality that it will be available for use 24/7, while a human agent has, inherently, a limited time of work hours available to assist the customer. The customer will also be able to use the service on different channels like FB Messenger, Skype, or even in a webchat in the website and mobile app. This availability is a big benefit for the customer since he won't be too limited on how or where he can interact with the Chatbot.

In terms of sacrifices a customer may have to make when using this service, will be the loss of the personalized assistance that only a human is capable of. The service will also not be able to solve too complex issues, those that are too particular to a specific case, and therefore cannot be trained by the developer for the Chatbot to understand. The customer may also initially not trust the service, being more used to have its issues handled by humans, and therefore being wary as to whether the Chatbot will be able to solve his issues or not. And in cases in which the customer needs to be redirect to a human agent due to the complexity of the issue to be solved, he or she may get the sense that the time spent using the service was time wasted.

One other sacrifice that the customer may encounter is the possibility that the service will have bugs unknown to the developer, failing to correctly solve the issue at hand, thinking this is not something that will happen when interacting with a human agent.

However, although these sacrifices are apparent, one benefit to the Chatbot is that it would not exclusively be the Customer Service type available. The customer would still be able to talk to a human agent, either by being automatically forwarded by the Chatbot when it understands it needs to, or by simply choosing to do so.

At the end of using the service the customer will be able to evaluate its performance, giving the developer precious feedback that will enable to further improve the Chatbot so that it will become better with time.

2.2.2 Value Temporal Position

Woodall (2003) divides Value for the Customer (VC) into four value temporal positions. Ex Ante VC (pre-purchase), Transaction VC (at the point of trade), Ex Post VC (post-purchase) and Disposition VC (after use/experience). In this section we will divide the benefits and sacrifices that the customer will have with the Chatbot into these positions.

- Ex Ante VC
  - Benefits
    - Time and Channel's Availability
  - Sacrifices
    - Mistrust
    - Unfamiliarity
- Transaction VC
  - Benefits
    - Simplicity
    - Effectiveness
  - Sacrifices
    - Inability to solve Complex Issues
    - Possibility of encountering software bugs
- Ex Post VC
  - Benefits
    - Issue Solved
  - Sacrifices
    - Time wasted with service if redirected
- Disposition VC
  - Benefits
    - Possibility of evaluating service

2.3 Value Proposition

The Value Propositions of this project are the easy service management that will be possible to do when using the Chatbot developed, either creating, modifying or cancelling services, and the quick and effective customer assistance. This will be because of the possibility of the Chatbot handling all the simpler issues that customers or potential customers may have with the marketplace, leaving the human agents to deal with the bigger and more complex ones. And since the Chatbot will be available 24/7 and on multiple channels, answering the users almost immediately, the issues it will be able to solve are bound to be more effective than if the user had to contact an agent, within the hours it would be available at work.

2.4 Canvas Model

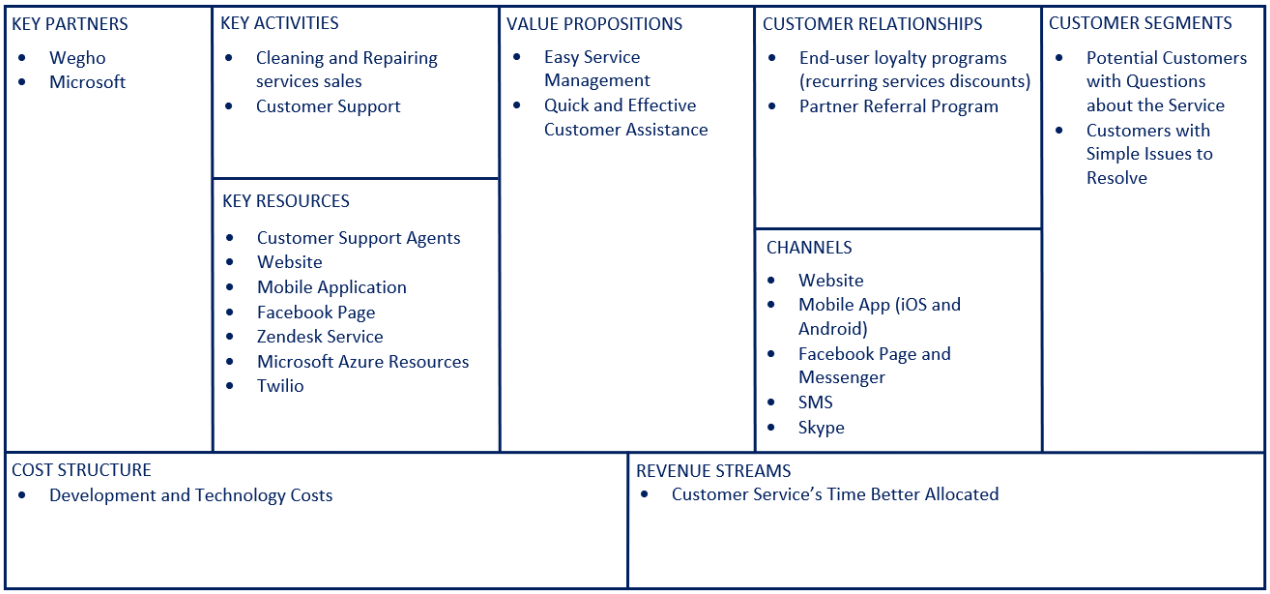

Figure 2 – Business CANVAS Model

The Business Canvas Model for this project will be explained in this section and can be seen illustrated in Figure 2.

The Key Partners on this project are Wegho, the product owners; B2F, the company who developed both the website and mobile applications; and Microsoft, whose resources will be used to develop the Chatbot.

The Key Activities of the project are Repairing and Cleaning services sales, which will be scheduled with the help of the Chatbot developed on this project, and Customer Support, which will be supported through the help that this solution will provide in answering client's issues.

The Key Resources the marketplace and the Chatbot will have are the Customer Support Agents, who handle all the issues required and who will work coordinating with the Chatbot, taking care of the issues the Chatbot cannot deal with. This coordination will be present with the creation of tickets on the CRM Zendesk Service, and the Chatbot will be present on multiple channels, being the FB Messenger (and through the Customer Chat Plugin also on the website and the mobile apps). The Twilio platform will also be used to communicate with the service providers through SMS if required. The Chatbot will be developed by using the Microsoft Azure Resources, such as the Bot Framework, QnA Maker and LUIS.

As Customer Relationships, the customers will have discounts when using the services that the marketplace has to offer repeatedly, having the price shortened when scheduling on monthly occurring services for example. And every time a user invites another user, both will receive 10€ credit to use when scheduling a new service.

The Customer Segments who will use the Chatbot service will be customers, or potential customers, who have questions about the Wegho Marketplace or who have simple issues to be done, without having the need to use a human Customer Support agent.

The Cost Structure of the project to be developed will be only the costs associated to the development of the project, such as the working hours and the prices of the resources to be used. The Revenue Stream will come from the human agents of the Customer Support having their time better allocated, not having to spend so much time handling simpler and ordinary issues that can be solved by the Chatbot.

3 State of the Art

In this section several Chatbot creation tools available for use are studied and all their characteristics are analysed so that a coherent and significant evaluation of the choice of which to use can be made.

3.1 Microsoft's Azure Bot Framework

The Microsoft's Azure Bot Service provides an integrated environment that is purpose-built for bot development, enabling the building, connecting, testing, deployment and management of intelligent bots, all from one place. Bot Service leverages the Bot Builder SDK with support for .NET and Node.js (Microsoft Azure, 2017 a). It also offers a comprehensive documentation area, full of get started guides, tutorials and samples.

The Bot Service also comes with easy configuration to work on multiple channels, including FB Messenger, Skype, Skype For Business, Microsoft Teams, Slack, Twilio, Kik, Telegram, SMS, Cortana, email, and in a website or app through a webchat component.

During the development it's possible to test the Chatbot using the Bot Framework Emulator, allowing to chat with it, testing all the coded functionalities, without the need of previous publishing.

The Bot Framework comes with two pricing tiers to be chosen. Both tiers allow for free unlimited messages on so-called standard channels, such as Skype, FB Messenger, Slack, among others. But while the Free tier limits the number of messages to be sent on the Premium Channels, like a Web Chat component, to 10.000 per month, the Premium Tier has no such limit, costing €0.422 per 1.000 messages.

Apart from the service cost, the Chatbot will also be powered by Azure App Services and will have the additional costs for the resources consumed, as described in Table 1.

Table 1 - Azure App Service Pricing Details

	FREE	SHARED	BASIC	STANDARD	PREMIUM	ISOLATED
Web, Mobile or API Apps	10	100	Unlimited	Unlimited	Unlimited	Unlimited
Disk Space	1 GB	1 GB	10 GB	50 GB	250 GB	1 TB
Maximum Instances	-	-	Up to 3	Up to 10	Up to 20	Up to 100
Custom Domain	-	Supported	Supported	Supported	Supported	Supported
Auto Scale	-	-	-	Supported	Supported	Supported
VPN hybrid connectivity	-	-	-	Supported	Supported	Supported
Network Isolation	-	-	-	-	-	Supported
Price per Hour	Free	€0.011	€0.064	€0.085	€0.085	€0.338

A benefit of using the Bot Framework is its connectivity to other Azure Services, more specifically the Cognitive Services. These are a set of services that add intelligence to applications and include LUIS (Language Understanding Intelligent Service) and QnA Maker, among others.

3.1.1 LUIS

LUIS is a machine learning-based service, used to build natural language into apps, bots and IoT devices. This works by defining the intents and entities the service will have to comprehend from the user's sentences. Intents are the goals the user might have by what have through what he says (e.g., create a service, login, etc…) and the entities are valuable information to be gathered from the sentence (e.g., date, address, name, etc…). (Microsoft Azure, 2017 b)

LUIS will be able to identify such intents and entities by being trained to do so. The developer will have to define everything that will be needed to be extracted from the user's sentences, by means of giving examples of utterances containing such elements. The service will then use machine learning to train itself with these utterances and, after being published, use active learning to improve identification. In this process LUIS will also provide the developer with real utterances that it is relatively unsure of for review. (Microsoft Azure, 2017 b)

After the model is designed, trained and published, it receives utterances through HTTP requests and responds with the extracted user intentions as a JSON object. These responses include a score for each element identified, which represents the percentage of which LUIS is certain that it belongs to the intent or entity required. A representation of this process is shown in Figure 3

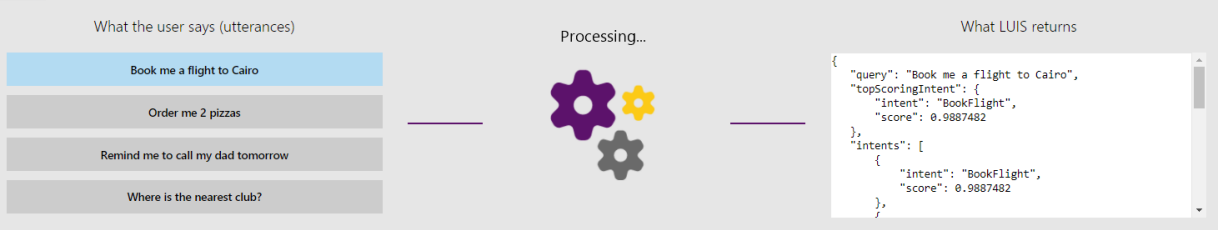

Figure 3 - LUIS Response Example

The LUIS service, like the Bot Framework, also has two pricing tiers to choose from, the free and basic tiers, as described in Table 2.

Table 2 - LUIS Service Pricing Details

TIER	TRANSACTIONS PER SECOND	PRICE
Free	5	10,000 transactions free per month
Basic	50	€1.265 per 1,000 transactions

3.1.2 QnA Maker

The QnA Maker is a REST API and web-based service provided by Microsoft that utilizes a trained knowledge base to understand natural language sentences, corresponding them to the context, and that tries to give the respective answer. (Microsoft Azure, 2018 g) The knowledge base can be trained through different sources, either by linking to an existing frequently asked questions page on a website, which then translates to question and answer pairs, by giving a structured file containing the pairs, or by manually providing it individually with each pair.

After the knowledge base is trained, an application can send a POST request to the web-service endpoint, which will then respond with a JSON with the list of all possible answers, sorted by their ranking score. This score is the percentage of which the service is certain that the answer corresponds to the question asked.

3.2 Watson Conversation Service

The Watson Conversation Service is IBM's Chatbot creation service. It combines machine learning, natural language understanding and integrated dialog tools to create conversation flows between applications and users.

When the service receives user input, it connects to a workspace, a container for the dialog flow and training data. Then it interprets the input, directs the flow of the conversation and gathers the information. Afterwards the application interacts with the developer's back-end systems, based on the information gathered, and functions as required (IBM, 2018 b).

The training data, just like on Microsoft's framework, consists of intents and entities and uses sample utterances to be trained. And as training data is added, a natural language classifier is automatically added to the workspace and is trained to understand the types of requests indicated and that the service should listen to and respond to.

The service uses a graphical representation of the dialogs' flows, giving the developer the option to add branches to each of the intents to be handled. Each branch can then have nodes that handle the many permutations of a request, based on the entities found or other information passed to the service from the application or another external service.

The application can also use other IBM Watson services such as Tone Analyzer, Speech to Text or Text to Speech to increase the functionalities of the Chatbot. The first understands emotions from the user's sentences, and the other two services can convert the utterances said from text to audio, and vice-versa (IBM, 2018 a).

A representation of this service's architecture can be seen in Figure 4.

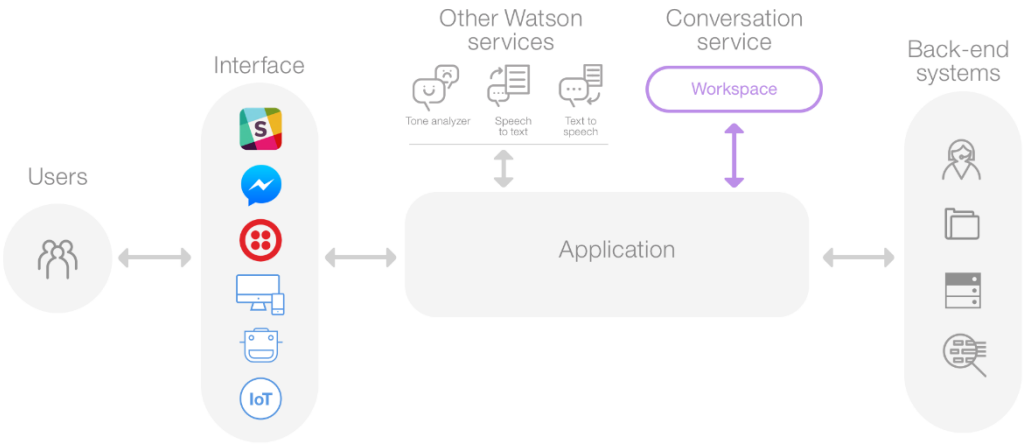

Figure 4 - Watson Conversation Solution Architecture

Watson's Conversation Service has two pricing tiers to choose from as to work with, the Lite and Standard, and are as described in Table 3.

Table 3 - Watson Conversation Service Pricing Details

	LITE	**STANDARD**
API Calls	Up to 10,000 API calls	Unlimited API queries/month
Workspaces	Up to 5 workspaces	Up to 20 workspaces
Intents	Up to 100 intents	Up to 2000 intents
Entities	Up to 25 entities	Up to 1000 entities
Cloud	-	Shared public cloud
Price	$0 USD	$0.0025 (USD) per API call

3.3 Chatfuel

Chatfuel is a Chatbot development service that focuses on the creation of bots without the need of programming. It uses Artificial Intelligence to recognize phrases sent by a user and replying the defined relevant answer. It also has no cost attached to working with it (Chatfuel, 2018).

The developer can train the AI by creating “AI Rules”, pairs of multiple possibilities of questions or simple sentences a user may send to the Chatbot, and the appropriate answers to reply with. These replies can be simple text phrases, or previously set up text blocks that can be connected more than one AI Rule. These Rules can also be structured in AI Groups, so as to have a better organization. An example of these rules on the Chatfuel website can be seen in Figure 5.

Every time a user sends the Chatbot a sentence that Chatfuel cannot correspond to an existing rule, it is saved in a section called “User Inputs Not Recognized By AI”, saving also the number of times that specific sentence was asked. The developer can then use the sentences in that section to better train the Chatbot, adding them to the corresponding rule.

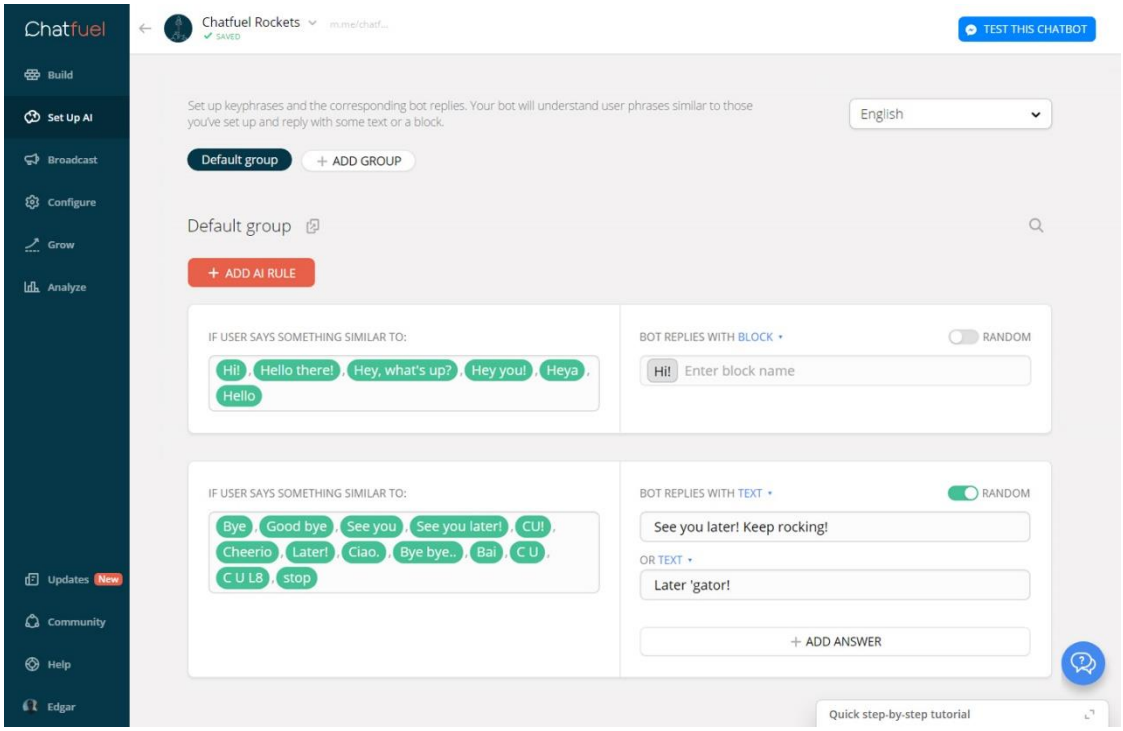

Figure 5 - Chatfuel AI Rules Example

Chatfuel uses FB Messenger as its channel of distribution. To create a new Chatbot with this tool, the developer must either associate it with an existing page or create a new one. After being setup, every message that is sent to that page will be directed to the Chatbot.

This tool also has the functionality of broadcasting messages to its users. These are messages, with text only or with more objects such as buttons or an image, that can be sent to all users that have previously talked with the Chatbot. These can be sent immediately or be scheduled

to be sent at a specific time or as a sequence, a response to one or more predefined text blocks, sent after a defined set of time.

3.4 Comparison

When comparing the studied tools, what can first be seen is the difference in working with Azure's Bot Service and Watson's Conversation Service or Chatfuel. The first allows for a more complex development, using programming to define the flows of dialogs, while the other two services prefer a more graphic and user-friendly development. And while Chatfuel is exclusively to be used with the FB Messenger channel, Watson's service allows for more channels and Azure's allows for even more, having the widest range of available services. Also, whereas Chatfuel has no cost to work with it, both the other services have prices, divided in different tiers from which to choose from, being Azure's the more expensive one, especially because it asks for payment for the main service, plus LUIS cognitive service and for the Azure App Services' resources consumed. This tools' comparison can also be seen represented in the Table 4.

Table 4 - Chatbot Creation Tools Comparison Table

Tool	**License**	**Channels**	**Portuguese Available**	**Programming Languages**	**Tools Integrated**
Microsoft Bot Framework	Free Basic	Skype, FB Messenger, Email, Twilio, Microsoft Teams, Slack, Web Chat	Yes	C# SDK Node.js SDK	Azure Cognitive Services
IBM Watson Conversation	Lite Standard	Slack, FB Messenger	Yes	Node SDK Java SDK Python SDK iOS SDK Unity SDK	Watson Services
Chatfuel	Free	FB Messenger	Yes	Not available	-

4 Analysis

In this section an analysis of the project to be developed will be made to fulfil the objectives defined in Section 1.3. A Use Cases diagram (Figure 6) will be used to illustrate and explain the different types of users that will use the final product, and what actions each will have.

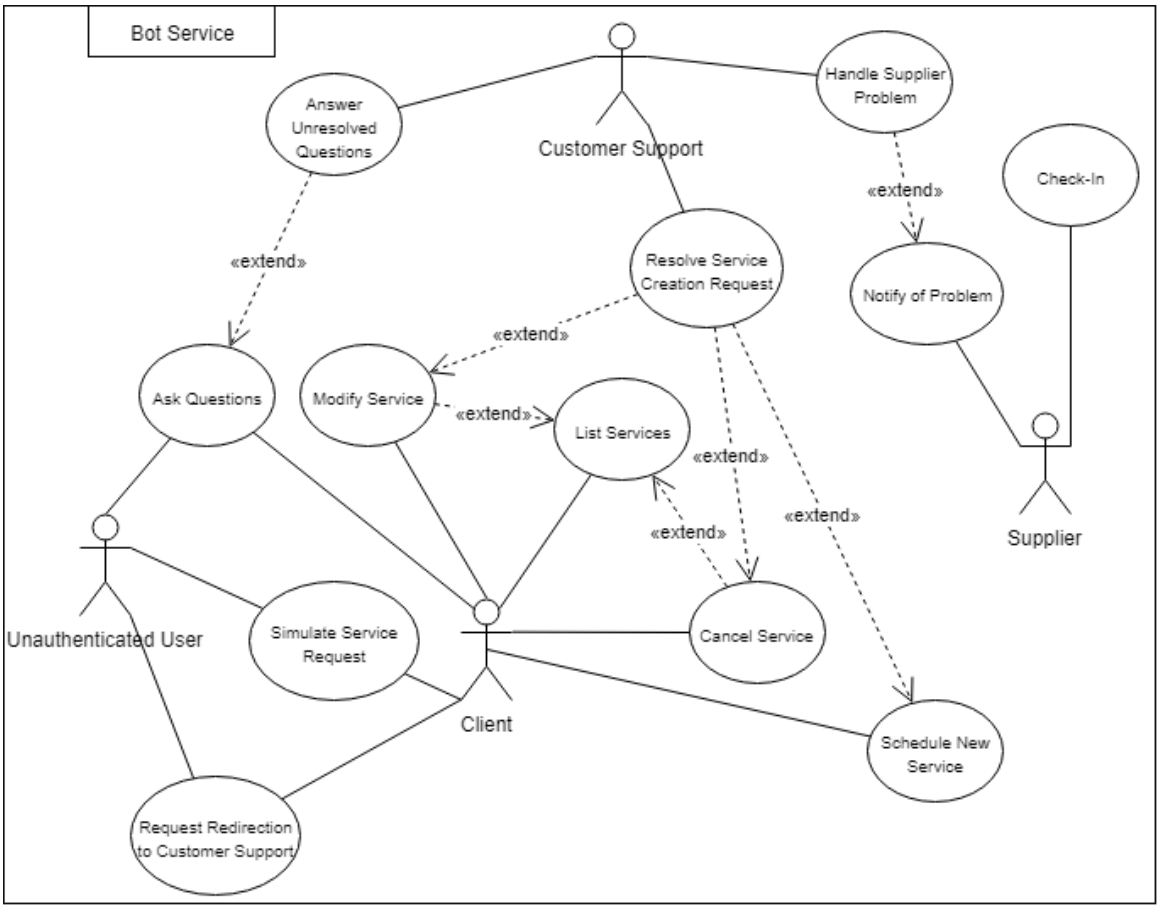

Figure 6 - Use Case Diagram

A description of the non-functional requirements that the project will have to fulfil will also be detailed.

4.1 Actors

There are four actors who will interact with the solution, the Client, the Unauthenticated User, the Supplier and the Customer Support.

The Unauthenticated User will be able to simulate new service requests, request to be redirected to the Customer Support and ask questions to the Chatbot.

The Client will have the same capabilities of the Unauthenticated User, plus the possibility of managing services, which includes the creation of new ones and the modification of those that already exist, either by making a change request or cancelling it. They will also be able to ask questions to the Chatbot, which if it can't respond to will ask if they want to be redirected to the Customer Support. The Client can also directly request this at any point during their interaction with the Chatbot.

The Customer Support will receive the service management requests and act accordingly to them. They will also receive the questions that were asked and that the Chatbot was not able to single-handedly respond to.

The Supplier will be notified when he hasn't checked in on a service after 15 minutes of its scheduled starting time. If he is delayed by any circumstance or has had an issue that has inhibited him to be on the location of the service, he may directly respond to that notification explaining the reason for this.

4.2 Use Case Details

All the use cases will be explained in detail here, describing what will be required of each.

4.2.1 Ask Questions

When a user asks questions to the Chatbot it will have to be able to respond with a concise and correct response. The actors that will be able to do this use case are the Client and the Unauthenticated User.

4.2.2 Simulate Service Request

Both the Client and the Unauthenticated User actors will be able to simulate the creation of new services. They will give details for that service and will receive a response from the Chatbot containing whether or not it would be able to schedule and the price it would have.

4.2.3 Schedule New Service

When a user requests the Chatbot to create a new service, this will follow the same procedure of the Simulate Service Request use case, except that at the end it will allow for the saving of the request. This use case will only available to the Client actor.

4.2.4 Request Redirection to Customer Support

A user will be able to request to be redirected to the Customer Support at any time during his use of the Chatbot. This can happen because it wasn't being able to correspond to its questions or demands, or if the user explicitly request this. This use case will be available to the Client and Unauthenticated actors.

4.2.5 List Services

Users will have the capability of request to see their previously scheduled services, no matter what platform they used to schedule them. This request can be given by giving a specific date or date span of the services which they want to see, or not, to which the Chatbot will list all the services scheduled for the following year. After listing them the Chatbot will give the user the possibility to do multiple actions to any one service, which includes cancelling or modifying it. This use case will only be available to the Client actor.

4.2.6 Modify Service

Users of the Client actor will also be able to modify their previously scheduled services, either by requesting to do so or after they list them. After asking to do so the Chatbot will ask the user what modification they want to make and will then save the request for further handling by the Customer Support.

4.2.7 Cancel Service

Users will also have the ability to request the cancellation of services. Similar to the Modify Service use case, this can be requested explicitly or after the listing of multiple services. After a confirmation from the user the Chatbot will save the request, which will also be handled by the Customer Support afterwards. This use case is also only available to the Client actor.

4.2.8 Answer Unresolved Questions

This use case is available to the Customer Support actor and it involves answering questions that were made by other users to the Chatbot (4.2.1) that it could not give an answer to.

4.2.9 Resolve Service Management Request

The Customer Support actor will also have the capacity to resolve service management requests. These involve the Client actor's requests to create, modify or cancel a service, to which the user will have to associate that request on the Wegho platform.

4.2.10 Notify of Problem

The Supplier actor will be able to notify of a problem or any situation that impeded him to check-in on time. This notification will be made through a SMS message in response to one that he has received and will be forwarded to the Customer Support as an e-mail.

4.2.11 Check-In

The Supplier will also be able to check-in to a service in case that he received a notification that he missed the check-in deadline due to forgetting to do it.

4.2.12 Handle Supplier Problem

The Customer Support actor will have also the capacity to handle the Supplier actor's problem notification that's explained in the Section 4.2.10, talking with the respective service's client if necessary.

4.3 Non-functional Requirements

The Chatbot will also have to be able to correspond to various non-functional requirements, which will be described in this section.

4.3.1 Integration with Zendesk

The Chatbot will need to be integrated with Zendesk in order to create customized support tickets to later be handled by the Customer Support.

4.3.2 Integration with Twilio

It will also be necessary for the Chatbot to be integrated with the Twilio platform for the sending and receiving of SMS messages for the Supplier actor's use cases functioning.

4.3.3 Integration with Wegho Web Services

The Wegho web service will also need to be integrated with the Chatbot for the execution of two main purposes: the authentication of a user, transforming him from an Unauthenticated User actor to a Client one; and for the validation of service details when simulating or creating one. This will include verifying every detail that the user gives to see if they are within the parameters set in the Wegho back-office.

4.3.4 Communicate with LUIS

The Chatbot will need to communicate with the LUIS service for the capability of understanding the natural language used by all users and catching their intents.

4.3.5 Communicate with QnA Maker

The Chatbot will also need to communicate with the QnA Maker service in order to search its knowledge base for the answer of a potential user question.

4.3.6 Notify Supplier

This project's end product will have to be able to know when a supplier doesn't check-in to a service 15 minutes after it was scheduled to begin, having then to notify the supplier of such event.

4.3.7 Redirect Supplier's notification's answer to Customer Support

Lastly the end product will have to be able to catch a Supplier's notification SMS answer and forward it to the Customer Support in form of an e-mail to be handled.

5 Design

In this section the Design of the project will be explained. First all the technologies used are enunciated, following with a representation of the classes constructed for the final product. The workflows designed for the execution of the use cases listed in Section 4.2 are also illustrated in this Section. Lastly, the architecture of the whole project can be seen designed and explained, including its representation through the use of a Components Diagram.

5.1 Technologies

The solution designed with the Microsoft Bot Framework was developed in a Visual Studio Project. It was constructed using the C# language and the Bot Builder SDK for .NET framework. This framework works with the Azure Bot Service to design and build the dialogs that make the Chatbot, using Language Understanding Intelligent Service (LUIS) and QnA Maker to understand the user's utterances and their intents.

5.1.1 LUIS

LUIS is a machine learning-based service to build natural language understanding into apps. It is designed to identify valuable information in conversations, interpreting user goals (Intents) and distilling valuable information from sentences (Entities). It uses reinforcement learning to continuously improve the quality of the natural language processing model. Once the model starts processing input, LUIS begins active learning, allowing whomever is managing the service to constantly update and improve it (Microsoft Azure, 2018 f).

When the model is ready and published the Chatbot can use it as a REST API, sending the user's utterances in JSON format and receiving in exchange another JSON which contains the Intent found to be the most appropriate and the Entities recognized, if any.

5.1.2 QnA Maker

QnA Maker is a web-based service that takes a knowledge base consisting of a set of question and answer pairs and builds a REST API that can be used to respond to user's questions. This

knowledge base can be extracted from an existing webpage or document, like for example a FAQ page on a website, or it can be built manually by adding all the questions and answers the API is supposed to be able to respond to. It can also be improved by adding more examples of how a question might be asked, so it is trained for more complex user queries. The more different examples of how to make a question, the better prepared it will be.

5.1.3 Azure Functions

A C# function was also developed using Azure Functions, to work as a proxy between the FB Messenger and the Chatbot. This function works as an HTTP-based API endpoint that is called every time a new message event should be sent to the Chatbot from the Messenger, getting, in the middle of the two so as to handle thread control issues.

5.1.4 Zendesk

Zendesk is a Customer Relationship Manager platform that works with the creation of support tickets and the handling of these by the Customer Support of the companies that use it. Using its API and the already existing usage of it by the Wegho company, the Chatbot creates tickets to notify the Customer Support that an action needs to be done by a human agent. This can be the finishing of a specific workflow or the help requested, due to lack of understanding or capability from the Chatbot.

5.1.5 Twilio

Twilio is a communication platform that, through an API, allows its users to send SMS messages to regular telephone numbers. The Chatbot uses this platform to implement the authentication of its users to their Wegho accounts, through the sending of a message with a randomly created token in order to verify the user's authenticity.

5.1.6 Azure Logic Apps

Azure Logic Apps is a cloud service that allows the creation of orchestrated and automated tasks, using multiple connectors such as HTTP, SQL, Twilio among many others. These can be triggered through these connectors, for example by receiving a HTTP POST request, or through scheduling. This service was used in two circumstances: to handle the verification of suppliers not checking

into services, and to save test issues in a database. These circumstances are explained in more detail in the Design Workflows (5.2.1) and Evaluation Methodology (7.3.2) sections respectively.

5.2 Class Diagrams

The class diagram of the Bot Builder solution can be seen in the Annexes in Section 10.1 in Figure 24. In it we can see all the classes created for this project. For this dissertation that class diagram was separated into two, one containing the main classes of the Bot Builder workflow. The other diagram includes all classes that the main ones will communicate with.

5.2.1 Main Classes

The main classes that regulate the workflow of the Chatbot are the Dialogs, which implement the IDialog interface. The first time a user interacts with the Chatbot, it will enter an instance of the RootDialog class, which will get the name of the user and introduce itself and its functionalities. Afterwards it will call the MainLUISDialog which will wait for a response from the LUIS API to know what the Intent of the user is so as to act accordingly. From this point on, the Chatbot, if a known Intent was found, will be redirected to the other Dialog classes, which are the following:

- CreateServiceDialog – Workflow to create a new service;
- CreateShortDialog – Workflow to get the minimum price of a service for a specific date and location;
- AuthenticateDialog – Workflow to authenticate the user to its Wegho account;
- ListServicesDialog – Workflow to list the user's already scheduled services and act on them if required;
- ManageServiceDialog – Workflow to manage a specific existing service, allowing the choice to cancel, modify or do nothing to it;
- CancelDialog – Workflow to cancel a previously existing service if requested and confirmed;
- ModifyServiceDialog – Workflow to ask the user the modifications requested for a specific existing service;
- CheckCityAvailability – Workflow that will check the user's given Zip Code to see if it's a location available for services.

A representation of these classes can be seen in Figure 7.

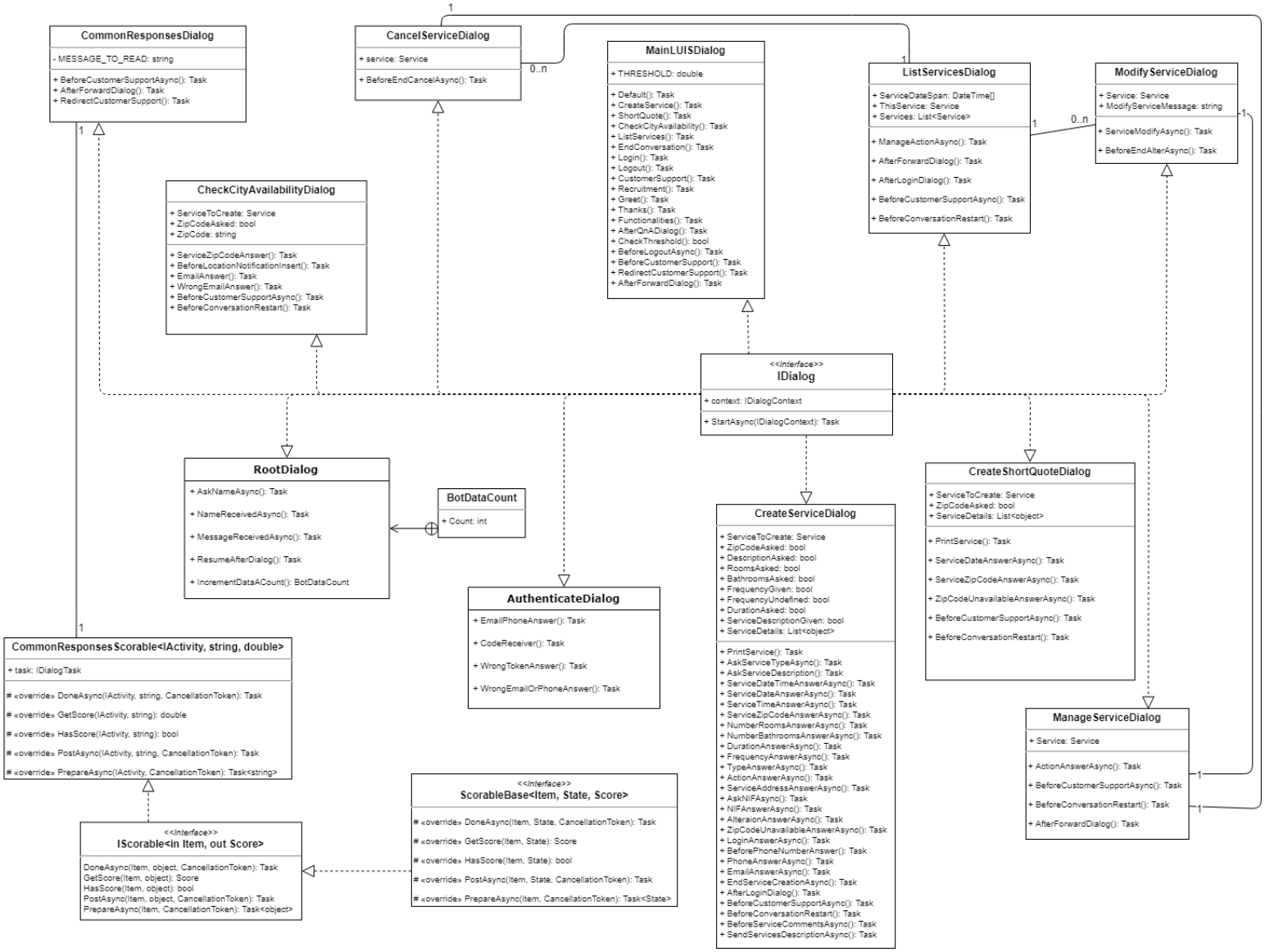

Figure 7 – Class Diagram of Dialog classes

5.2.2 Other Classes

The Dialog classes are complemented by two other types of classes. The first are the models, Service and QnAMakerAnswer. The Service is an object that is instantiated every time that a user requests to either schedule a new service or to view an existing one. The QnAMakerAnswer nests multiple classes: the QnAAnswer, the Metadata and the Answer. The QnAAnswer's only attribute is a list of Answer objects which, in each turn, will have a list of Metadata objects, besides other attributes. This class is instantiated any time that the Chatbot has to communicate with the QnA Maker API, being thus connected with most of the Dialogs.

The other class type is the Controller, which is all static classes where methods used multiple times inside other Controller classes and Dialog Classes are defined. These are represented in the class diagram in Figure 8 and are the following:

- DatesController – In this class all methods that interpret date values are defined, the main one being understanding DateTime values from various formats in sentences;
- QnAMakerController – This class is called any time a request to the QnA Maker API needs to be made. From that request it receives a JSON that is then deserialized into a QnAAnswer object, which contains all the answers found by the web service and returns the one chosen as the most appropriate;
- TicketsController – This class is where all the calls to the Zendesk API are made to create the different types of Tickets that may be required;
- WebServicesController – In this class, all the methods that call the Web Services described in Section 6.4.3 are implemented, in order to validate service details. It also includes the method called when a user requests to be redirected to the Customer Support and the method that sends a message to a given phone number through Twilio; ServicesController – In this class are all the other methods required for the logic of the Chatbot. This includes the retrieval of the user's Facebook name with the use of a request to the Graph API (Facebook, 2018 a), retrieving an integer from a user given sentence, and generating a random code to be sent to the user for authentication.

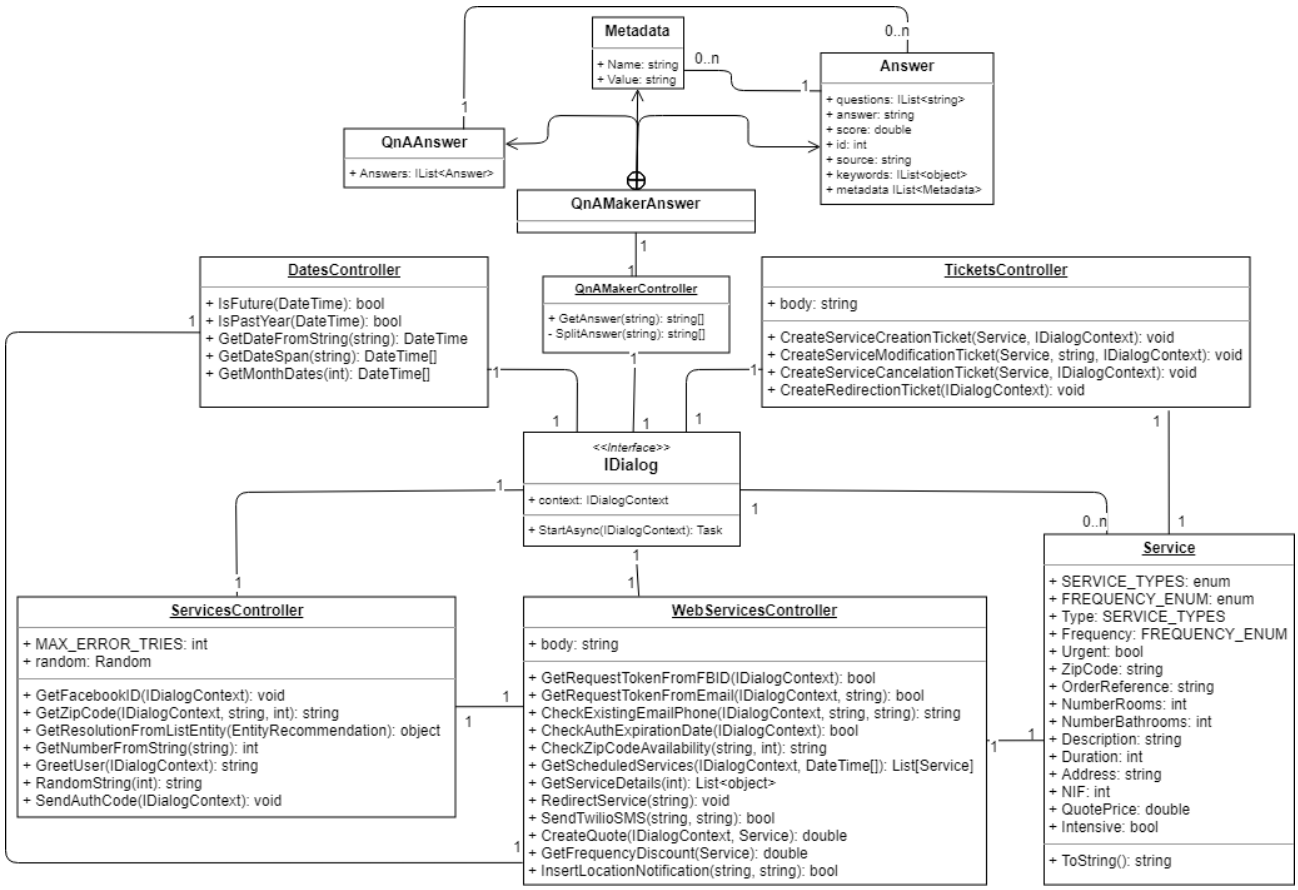

Figure 8 – Controller classes' Class Diagram

5.3 Design Workflows

From the Use Cases defined in the Analysis (Section 0), five main workflows were designed. User authentication, question and answer, service management, service creation, and also supplier's fail to check-in.

5.3.1 User Questions

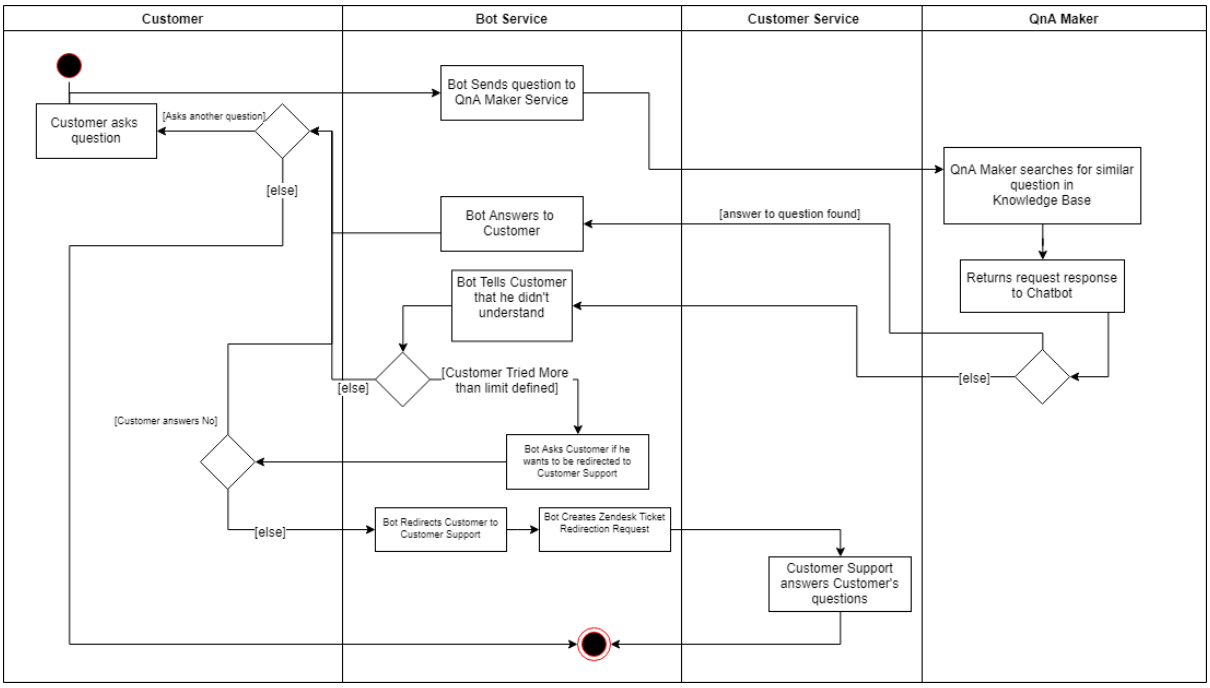

Figure 9 - User Questions Activity Diagram

As can be seen in activity diagram (Figure 9), every time a user asks the Chatbot a question, this will try and see if in the knowledge base of the QnA Maker service there is a question and respective answer that could be the solution to the user request. If one is found, it is returned to the user, after which he can decide to ask another question or not. If no solution is found in the knowledge base, then the Chatbot will notify the user of this. If this situation is repeated too many times, with its limit defined in the code of the Chatbot and with the possibility to be altered to a bigger or smaller number, the user will be asked if he wants to be redirected to a human Customer Support agent. If he chooses to do so, a Zendesk ticket will be created to notify the support of what has happened, and after which an agent can enter the conversation and talk with the user to satisfy his requests.

5.3.2 User Authentication

A user will enter the authentication workflow (Figure 10) any time he either requests to do so or is required because it is part of another workflow. When this happens, the Chatbot will ask the user to provide the e-mail address or telephone number that is associated with his Wegho account. If an account is found, the Chatbot will then use the Twilio service to send a message to the telephone number registered to it with a token to be returned by the user. If the user takes too much time to respond with the token, or gives an incorrect one, the Chatbot will ask if he wants to try again with a new token. If the user responds with the correct token, he will be authenticated and will have access to his scheduled services.

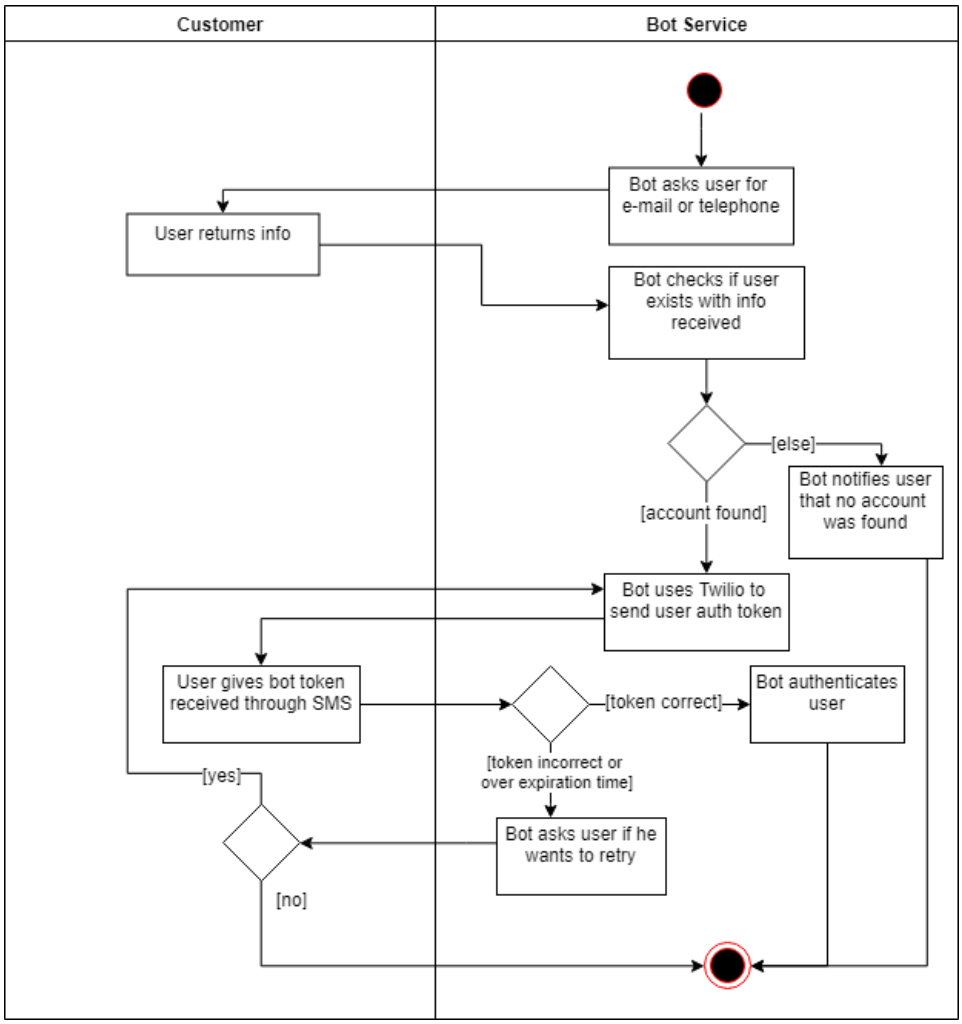

Figure 10 - Authentication Activity Diagram

5.3.3 Service Creation

When a user asks to create a new service the Chatbot will enter the respective workflow to correspond to this request, as can be seen in Figure 11. After analysing the request to retrieve all the information already provided by the user, the Chatbot will, if needed, ask the user for the remaining details. When all the details are received, the user will be shown a list of all the information passed including the quote price calculated, if possible, and will be asked to undertake an action. He will be able to alter any detail, cancel the request or confirm it. If he chooses to confirm it the Chatbot will then enter the authentication workflow previously explained in Section 5.3.2. After the success of this a Zendesk ticket will be created containing all the information of the service produced for the Customer Support to manage. At the same time the user will be notified by the Chatbot that his request was saved and will soon be handled, and that he will receive an e-mail confirmation after it.

The different types of services that the Chatbot will be prepared to offer are domestic and office cleaning services, all others being relegated to a simple “other” type, with a description field for the user to indicate what he may want. This was decided since all service types have different complexities to them and adding them all would exponentially increase the complexity and difficulty of the development. Thus, these two different types were selected since these are the most requested by customers.

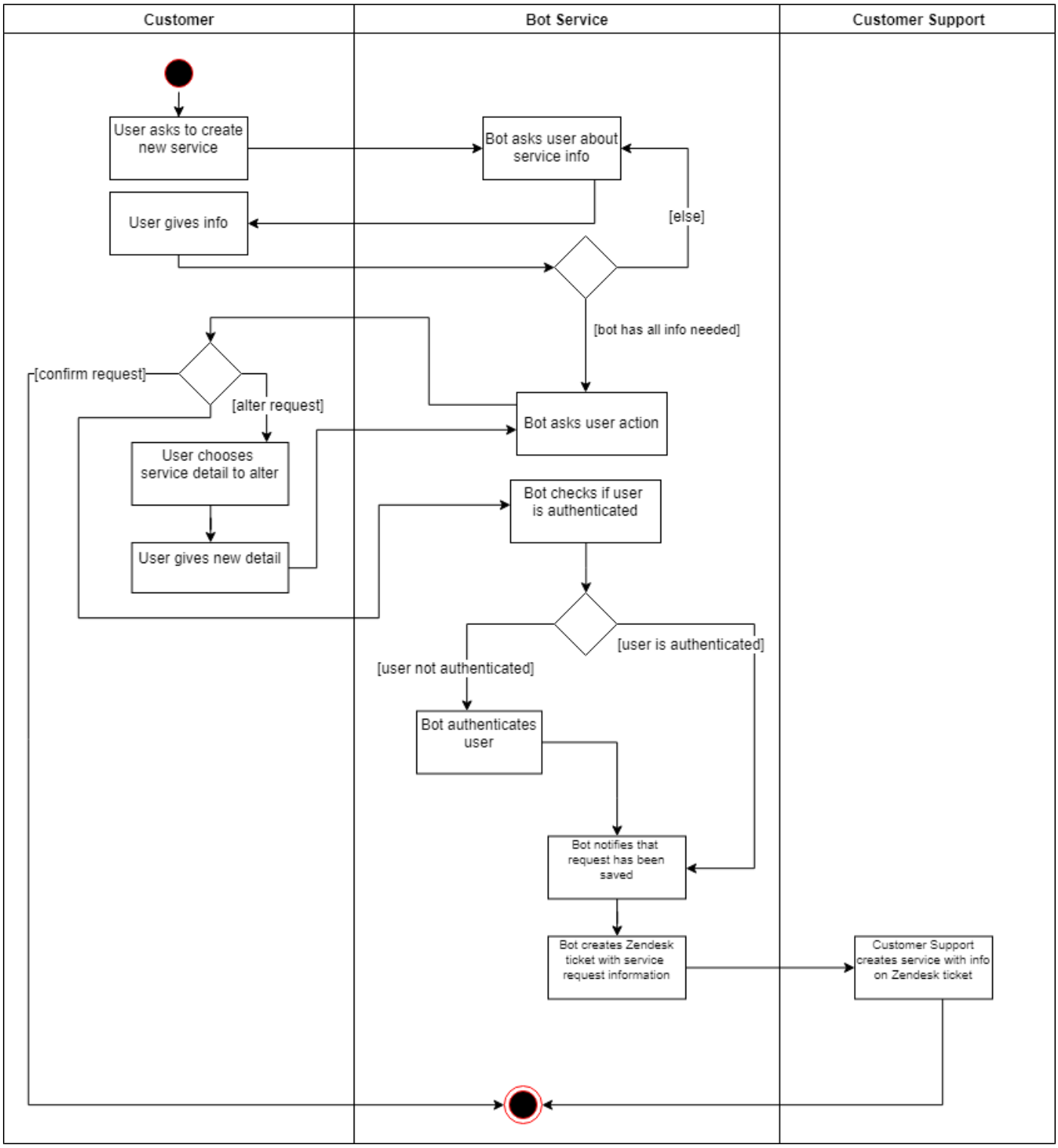

Figure 11 - New Service Creation Activity Diagram

5.3.4 Service Management

The next workflow is for when a user requests the Chatbot to modify, cancel or simply list the services that he has already scheduled. When this occurs, the Chatbot will first go into the authentication workflow described in section 5.3.2. After this the user will be shown a preview containing only the service type and date, restricting them to those in the date or date span requested, or in the following year if no such constraint was given. From these the user will be able to choose to take an action on any specific service, either to see more information about it, to cancel or modify it. After he chooses to see more information he will again be given the same choices of cancelling or modifying it, or to exit the workflow. If he chooses to cancel it, and confirms this request, a Zendesk ticket will be created for the Customer Support to do so, containing the identifying details of the service. If the user chooses to modify, it he will be asked to detail what he wants to have changed, so that the Chatbot can create a new Zendesk ticket with these details, or the Customer Support to properly handle. This workflow can be seen represented in Figure 12.

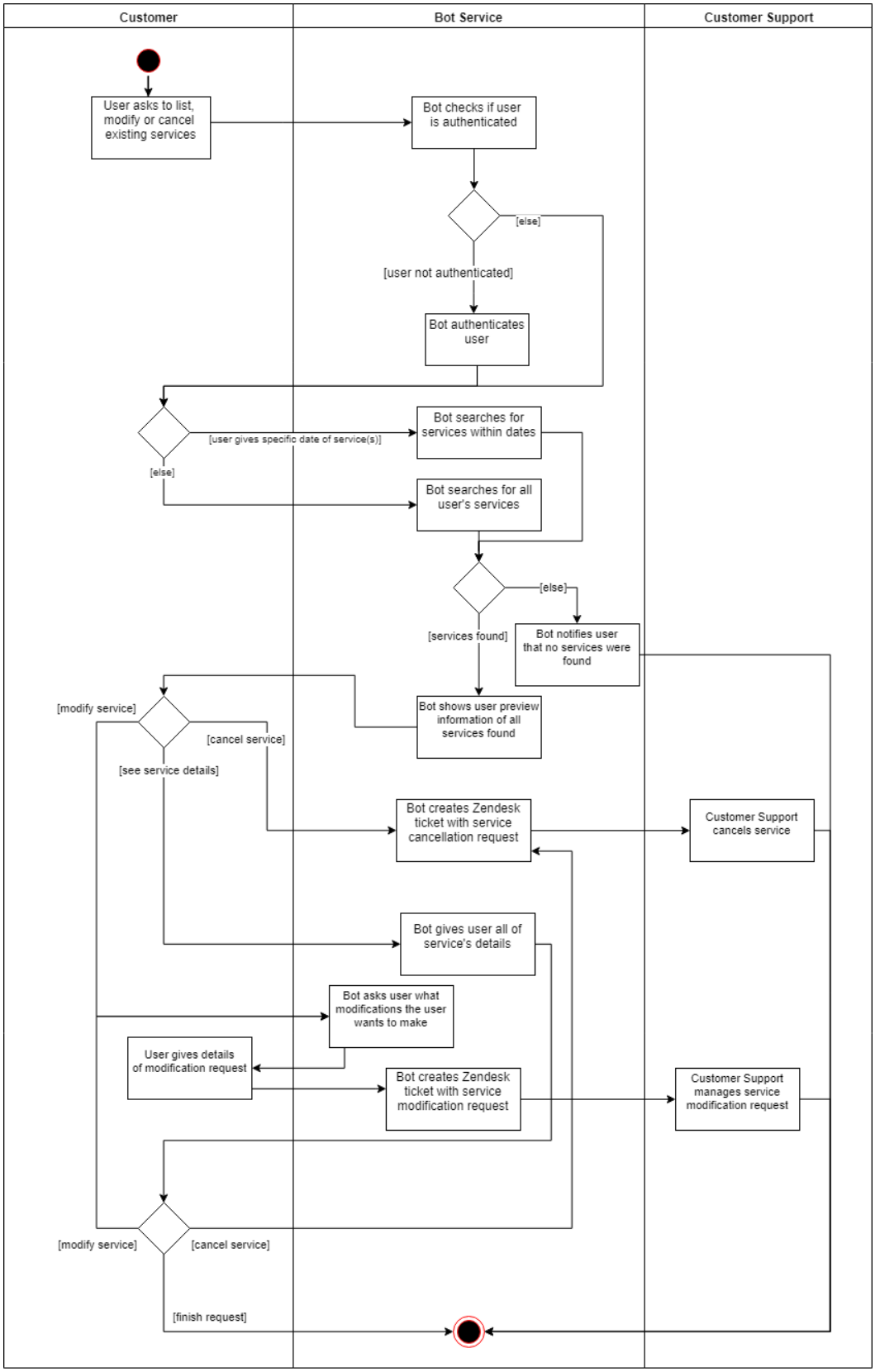

Figure 12 - Existing Service Management Activity Diagram

5.3.5 Supplier Check-In Verification

Every time a service supplier arrives at the location of a service he is supposed to check-in in the platform, so that the status of the service in the Wegho database is updated. If this doesn't happen, either when the supplier forgets to do so, or if he is delayed, 15 minutes after the service is supposed to happen, he will receive a SMS message notifying him.

This happens by having a Logic App running on a recurrence trigger, every 15 minutes. When it runs it executes a SQL query checking the status of every service scheduled between 15 and 29 minutes before the time of the trigger, acting on every instance whose status is not checked-in. When an instance like this is found the logic app will use a Twilio connector to send a SMS to the supplier, telling him that he is late in checking in. The logic app will only act on the services between these periods of time so that no supplier is spammed with multiple messages, receiving only one.

After a Supplier receives the message he is supposed to check-in, when the problem was that he forgot to do so when he arrived at the location of the service, or if he deems it necessary to respond to it, giving an explanation, for example saying that he will arrive even later than was expected and why. This answer will then be directed to the Customer Support as an e-mail that will act accordingly to it, notifying the Client if they decide that that is warranted to be done.

This workflow is represented in Figure 13.

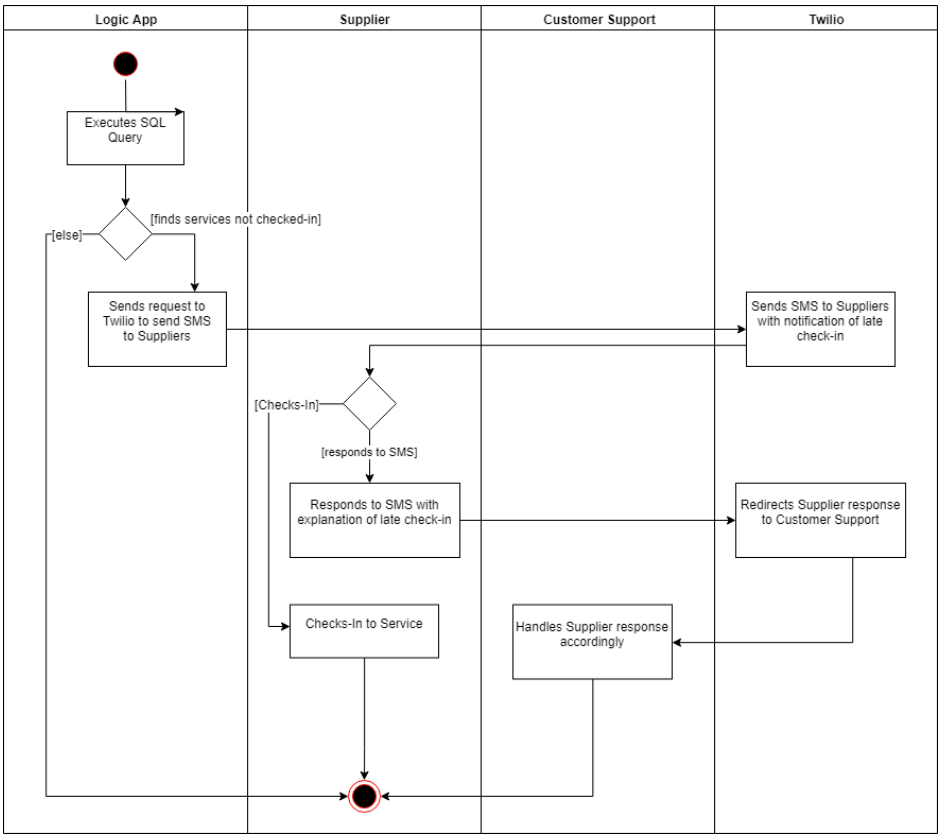

Figure 13 – Supplier Check-In Verification Activity Diagram

5.4 Design Alternatives

5.4.1 Use of Web Services instead of Zendesk Support Tickets

One alternative to the resolution of the cancelling and the creating of services workflows was to automatically solve them using webservices, instead of finishing them with the creation of a Zendesk ticket. Thus, when a user requested to cancel a service, at the end of the workflow the Chatbot would call a webservice that would set the service status in the Wegho database as cancelled. And when a user finished a new service request the Chatbot would call a webservice that would create it in the database. Immediately after this, the user would be able to check these changes in effect by asking the Chatbot to see the services, or to check them in the Wegho platforms.

This alternative was discarded due to the structure of the already existing back office of the Wegho platform. All the actions that were supposed to happen directly are currently connected to user sessions on the website and mobile apps, as to implement this system on the Chatbot would require a reformulation of how everything is currently working, including the authentication system. Consequently, it was decided that a simpler approach would be constructed for the Chatbot, creating the support tickets to be manually handled by the Customer Support agents.

5.4.2 Supplier Check-In

An alternative to the solution of the Supplier late check-in was to have the Chatbot directly send a notification to him, giving him the option to either check-in, inform of the estimated time of arrival to the location of the scheduled service, or ask to talk directly to the Client. This was to be handled through SMS messages, using the Twilio channel of the Bot Service. Eventually this solution was discarded because every message sent would have a price associated with it and having dialogs between the Supplier and the Chatbot on that channel would end up having too big of a cost for the simplicity and use of it. It would also be complex to implement with the Chatbot on how to automatically start, due to its nature of responding to User's requests, never starting preemptively.

Therefore, the use of a Logic App was decided to be the better solution due to its simple construction and because it would then be simpler to orchestrate when this workflow would happen using a Recurrence trigger and the execution of a SQL query. Also, through the sending of a single message to the Supplier, it was shown that it decreases the ultimate cost of this workflow.

5.4.3 Facebook Messenger as Exclusive Channel

The FB Messenger was chosen as the only channel on which the Chatbot would be currently available in contrast to what was defined on the objectives. This was decided because the platform has some advantages when using it exclusively. Firstly, it allows for its supervision to be made in a simpler way, having the Customer Support being able to see every conversation in real time on the Wegho's Facebook page.

It was also decided because Messenger already has a protocol to change thread control of the conversation between different apps, the Handover Protocol (Facebook, 2018 b). With this protocol the control of the conversation can be passed between the Chatbot and the Customer Support (in the Inbox Messenger app) in a simple and efficient way. This passage of control can be made by having a HTTP web request made with the logic code of the Chatbot, or by having the Customer Support manually doing it on the Facebook page.

Finally, with the existence of the FB Customer Chat Plugin (Facebook, 2018 f) it was easy to integrate the Chatbot in the website by including the Facebook Javascript SDK in the page where the plugin will be rendered.

The Chatbot could also be available on other channels like Skype, Twilio, MS Teams, among others. Nevertheless, having these would increase the complexity of the future management of conversations, as with for example Twilio, that would add a cost to every message that would be sent and received.

5.5 Architecture

This project contains two main solutions: the Wegho Bot, the Chatbot itself, and the Bot State Entities. These are the state of each conversation a user will have with the Chatbot, saving every information required in a database in a SQL Server. The Bot Framework also communicates with the two Cognitive Services utilized, LUIS and QnA Maker. All these previously mentioned components of the project are part of the same azure services resource group.

Out of this group, the Chatbot will communicate with the Zendesk software as the way for the customer support to manage every issue that the Chatbot will have to redirect.

The Chatbot will be available on FB Messenger and using Customer Chat Plugin, a Webchat component will also be available on the website.

This architecture was designed and transformed into a Components Diagram, which shows all the different parts of the solution and how they are integrated between each other (Figure 14).

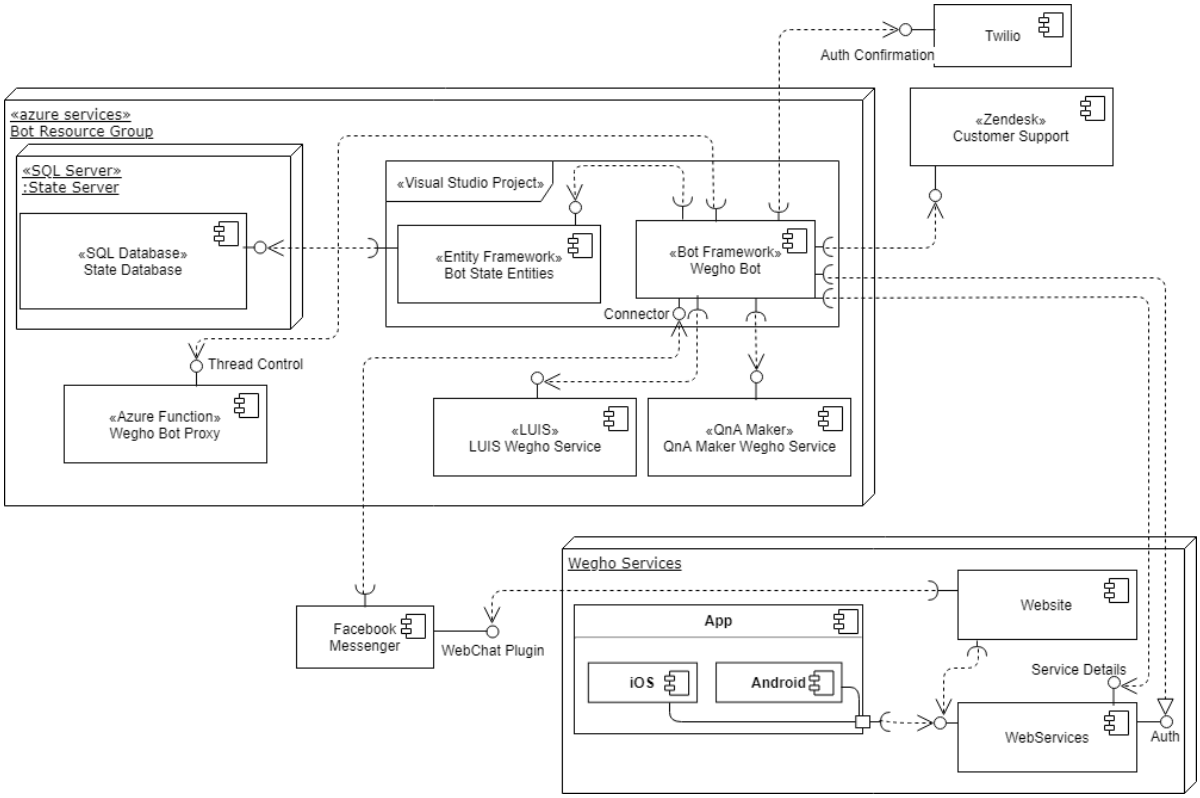

Figure 14 - Solution Architecture Components Diagram

6 Development

In this section, the development of the whole project will be explained, describing how each of its components was constructed and connected to each other. The main problems that occurred when developing them will also be discussed, how and why they happened, as well as what was done to solve them.

6.1 QnA Maker

To develop the QnA Maker service that would be called at any time a user would ask the Chatbot a question, it was necessary to provide it with the respective answers it would need to return, including multiple ways these questions could be asked. This was first done by giving it the URL of the already existing FAQ page on the Wegho website. With it, the service could automatically transform all the questions and answers that are there into the pairs that it would use.

After this was done, and was tested with the Chatbot, what could easily be seen is that the answers given, the ones that are on the website, are too big to be sent as messages. A user might ask a simple question of how to book a service, and receive a message with 500 words, making it very unattractive and in the context of a messaging platform not actually readable as people are accustomed to small concise messages when using these kinds of platforms.

Hence, what was decided to confront this issue was to automatically get the questions and answers from the website and to transform the answers into small condensed versions of them. These would contain only the essential information asked and at the end, give also the URL to the complete version on the website, so that if the user finds that he requires more, they could check there.

After all the questions and answers were defined, another step to improve the performance of this service is to give each of the pairs more examples of how a user might ask a given question. The bigger the quantity of different ways of asking a question, the more prepared the Chatbot will be to understand what the user's intentions are. An example of this can be seen in Figure 15.

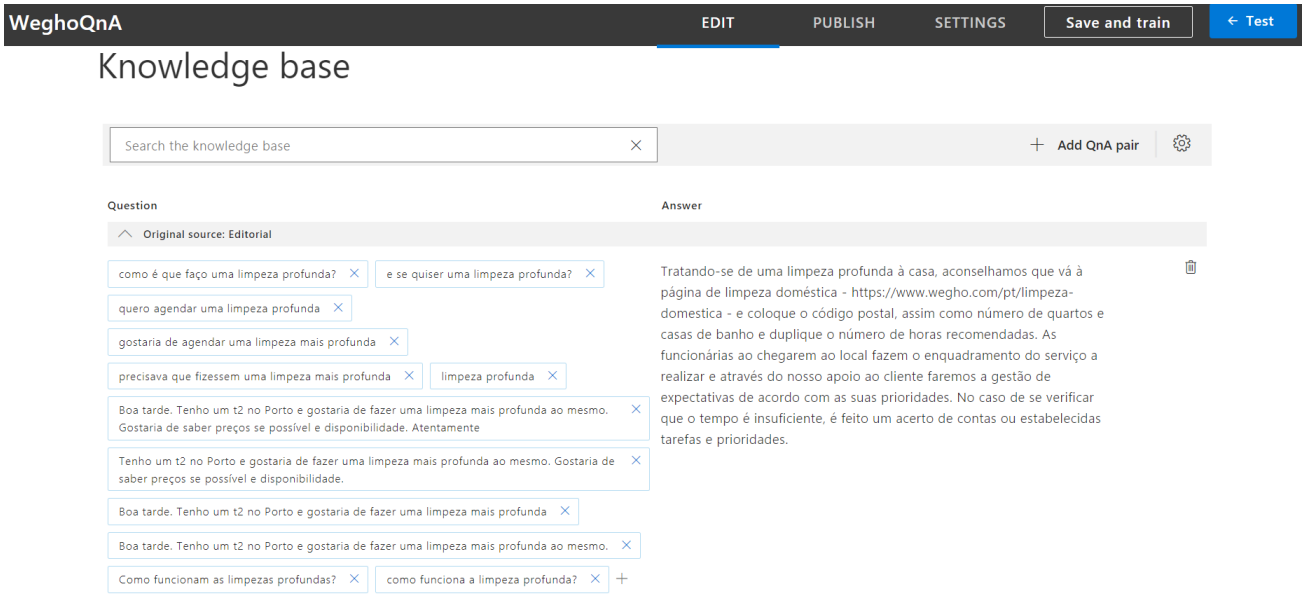

Figure 15 - QnA Maker Pair Example

6.2 LUIS

LUIS was used by giving it the main intents of the Chatbot to understand. These are all purposes defined that a user could use the Chatbot for, that aren't simply giving an answer. The exceptions to this rule are the "Recruitment", "Greet" and "Functionalities" intents. These three were chosen as exceptions because there are many different ways of giving these intents, and although they could be put in the QnA Maker knowledge base, LUIS has a functionality that's able to handle the association of new user utterances to an intent in a much easier way.

The Intents that were trained for the Chatbot to understand are the ones described in Table 5, and are complemented with the Entities listed in Table 6.

Table 5 - LUIS Intents

Intent	**Description**
None	Utterances with no own flow, either not designed questions or nonsensical words/phrases.
Service.CityAvailability	Utterances to direct to flow that checks if services can be done in given Zip Code.
Service.Create	Utterances to direct to creation of new service workflow.
Service.List	Utterances to direct to the listing of existing user services workflow.
Service.ShortQuote	Utterances to direct to workflow that gives minimum price of a service for a user given date and location.
Wegho.DirectCustomerSupport	Utterances to redirect user to a human agent of the Customer Support service.
Wegho.Functionalities	Utterances asking what are the Chatbot's functionalities and how to properly work with it.
Wegho.Greet	Greeting utterances, for which the Chatbot will respond with a greeting of its own.
Wegho.Login	Utterances that will start the workflow for the user to authenticate to its Wegho account.
Wegho.Logout	Utterances that will have the user log out of its Wegho account, deleting all account variables from the Chatbot context.
Wegho.Recruitment	Utterances for when a user asks if there is any recruitment open for a job in the Wegho company.

Each Intent is trained with utterances of how a user might ask for its specific workflow, which may or may not include instances of Entities. When creating a new Intent, the developer will give it some standard examples, trying to imagine the many different ways a user would ask for it. After receiving these examples, the API can be trained and published, updating its domain-

specific natural language model and improving its potential understanding of future user utterances, including both its intent and its Entities.

Table 6 - LUIS Entities

Entity	**Description**
Bathrooms	Number of bathrooms of service location.
Rooms	Number of rooms of service location.
Date	Date and Time intended for the service.
Duration	Approximate duration of service.
Frequency	Frequency of service, either unique, weekly, biweekly or monthly.
Intensive	If service is a deep cleaning, and not only a regular maintenance one.
Type.Cleaning.Domestic	Service requested is a domestic cleaning.
Type.Cleaning.Office	Service requested is an office cleaning.
Urgent	Service is urgent.
ZipCode	Zip Code of the requested service's location.

After these intents have primarily been defined, they can always be enhanced, providing more utterances in order to improve the natural language model. To do this the developer can go to the specific Intent in the LUIS webpage and add an utterance. If this added utterance includes instances of Entities, the developer should afterwards check if the model caught them and associate them in case it didn't. An example of this flow can be seen in Figure 16, where a new utterance is being added to the Service.ShortQuote, and then in Figure 17, where it can be seen that the model caught the Zip Code as an Entity.

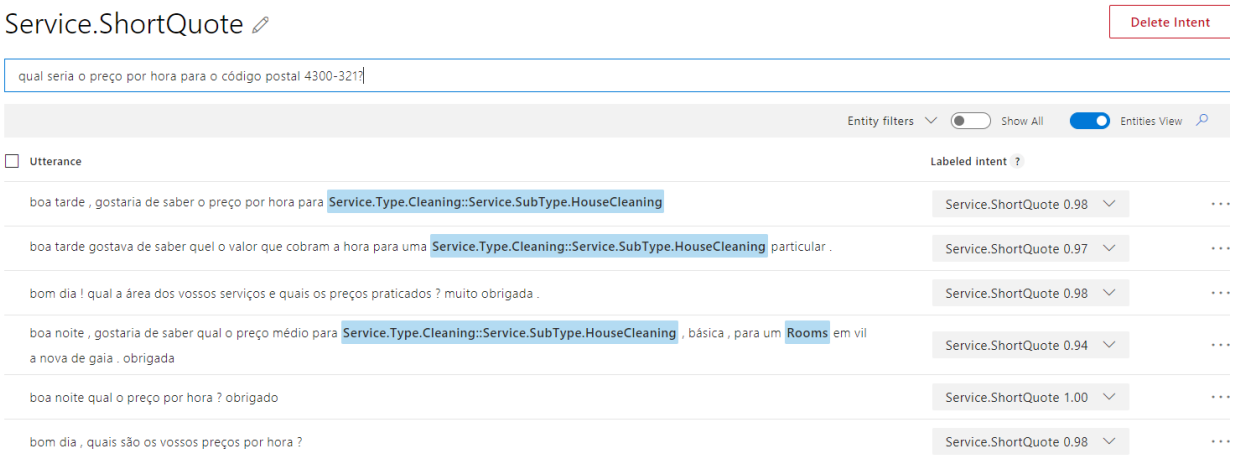

Figure 16 - Example of new utterance being added to LUIS Intent

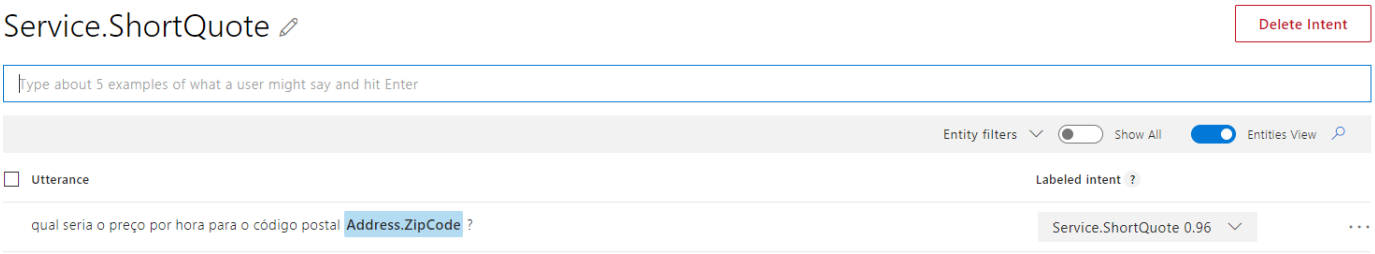

Figure 17 - Entity caught in new LUIS utterance

The LUIS website additionally includes a webpage for improving the model with user's utterances, titled "Review endpoint utterances" (Figure 18). This page presents to the developer every single utterance that was sent to the API and of which it wasn't 100% certain of its intention. The developer can then align it to the correct intent, checking for all the entities the sentence may have and saving it for the training and the re-publishing of the API.

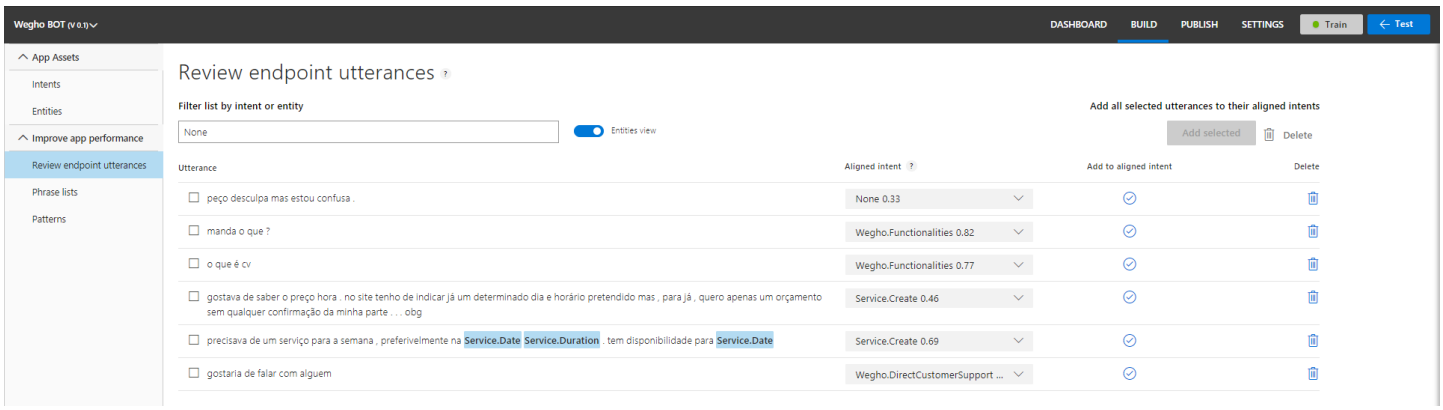

Figure 18 - Review endpoint utterances webpage

6.3 Facebook Messenger

FB Messenger was chosen as the sole channel on which the Chatbot would be available, as explained in Section 5.4.3, and to set it up a new Facebook App was created. In it Messenger was enabled and then associated with the Azure Bot service on Azure Portal. After the setup the Chatbot will be available for use to those who are certified to test the service (the product owner himself or a test user he approves). In order to make the Chatbot available to the those who need to interact with the Wegho Facebook Page, the app had to be previously submitted for review by Facebook (Microsoft Azure, 2018 d).

6.3.1 Persistent Menu

A persistent menu was added to the Chatbot with quick actions the user might choose to do at any time, interrupting the current flow of the conversation. These actions will always be available as side menu, also having the possibility of nested elements inside the menu. The actions available are the following:

- Services:
    - Create new service;
    - Cancel service;
    - Modify service;
    - List scheduled services;
- Help:
    - Functionalities;
    - Redirect to Customer Support;
- Back to start.

This menu was set up by using the Messenger Platform API, by sending a HTTP request with all the elements (Facebook, 2018 d) and an example of it can be seen in Figure 19. When an action is activated by the user the Chatbot logic starts a Scorable Dialog, as is described in Section 6.4.5.

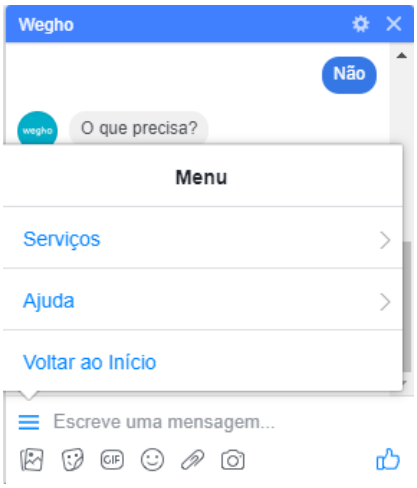

Figure 19 - Persistent Menu example

6.3.2 Get Started Button

Another element added to the Chatbot on the Messenger channel was the Get Started Button. This button is the first interaction a new user will always have with the Chatbot, after which it will respond with an introduction to the service, informing the user that he is talking with an artificial intelligence entity and indicating what it is capable of doing. This is set up by using the Messenger Platform API, sending a HTTP request and defining what the Chatbot logic will receive (Facebook, 2018 e).

6.3.3 Handover Protocol

The Handover Protocol was used to handle the handover of the conversation control between the Chatbot and the Customer Support. To set it up, the first thing was to go to the FB Messenger settings and defining the Chatbot App as the Primary Receiver and the Inbox as the Secondary Receiver. The Inbox is where all Messenger conversations usually end up and it's where the Customer Support is able to talk to the users with no interference from the Chatbot.

In order to handover the control between the two receivers, there were two possible ways of how to do it. One could be with a HTTP request to the Handover Protocol API (Facebook, 2018 c). The other way was to use the buttons automatically available for the Customer Support on the Facebook Page, “Send to Inbox” and “Done”. The first will appear when the Chatbot has thread control of the conversation and, if pressed, will request thread control which will be handled by the Proxy developed and described in Section 6.3.4. The second is present when the Customer Support has the thread control and, when pressed, will hand over the control back to the Chatbot.

6.3.4 Proxy

For the Handover Protocol to function as it should, a Proxy had to be developed to work between the Facebook and the Chatbot. This was due to the fact that when an agent of the Customer Support presses the “Move to Inbox” button in order to reclaim control of the conversation from the Chatbot, it simply requests the thread control (Section 6.6.3). Therefore, the solution was to develop a Proxy using an Azure Function in C# that would catch all the message events that were supposed to be sent from the Facebook to the Chatbot. This function will search in all messages for the keyword *request_thread_control*, which is sent when the “Move to Inbox” button is pressed. If this keyword is found, the proxy will then make a request to the Handover Protocol API, as described in Section 6.3.3, effectively handing over the control to the Customer Support. If this doesn’t happen, the proxy will just forward the message event to the Chatbot, resuming its normal flow. This solution was developed with the assistance of Gary Pretty, a Microsoft MVP, in a Bot Builder issue on its GitHub repository (GitHub, 2018 b).

In order to solve the problem later described on Section 6.6.1, another addition was made to the Proxy, which checks every message event, to check if the current thread owner is the Customer Support. Confirming it, the Proxy will ignore the event and won’t send it to the Chatbot as supposed to. This was happening due to the fact that the action on the persistent menu buttons would send the events to the Chatbot, disregarding who had thread control at the time. When the Bot Builder (Section 6.4.1) received the event, it did not have any information on who the thread owner was, this responding anyway.

6.4 Chatbot

The code of the Chatbot was developed using the Bot Builder SDK for .NET, a framework developed by Microsoft to function with the Azure Bot Service.

6.4.1 Bot Builder

The Bot Builder manages conversations through Dialogs. Dialog objects process inbound Activities and generate outbound responses. When at runtime, instances of these objects are organized in a stack, with the one on the top referred to as the active Dialog, which is persisted at each turn of the conversation.

A Dialog implements three main functions: BeginDialog, ContinueDialog and ResumeDialog. At runtime, Dialogs and the DialogContext class work together to choose the appropriate Dialog to handle an activity. This class ties the persisted stack, the inbound Activity and the DialogSet class. This last class contains all the Dialogs that the Chatbot can call.

Whenever there is no stack yet, the application will first begin the Dialog it chooses by calling the BeginDialog on the DialogContext, pushing the corresponding Dialog's id from the DialogSet to the stack. It will then delegate a call to BeginDialog on the specific Dialog object. If there is a stack, the application will then simply delegate the call to that Dialog's ContinueDialog, giving it any associated persisted properties. To support the nesting of Dialogs, by having parent and child dialogs, the DialogContext can call the ResumeDialog method on a parent Dialog when the child completes. (Microsoft Azure, 2018 e).

Inside the code of the Chatbot built with the SDK, the BeginDialog can be found at the top of every Dialog class constructed as the asynchronous method StartAsync. Afterwards, for every turn possible in the conversation there should be another asynchronous method where the logic of the Chatbot should be contained. Inside these methods the DialogContext is used to persist information about the conversation or the user, to send individual messages to the user that require no response, and to call other Dialogs. This calling can be done after asking the user some input, waiting for the response before continuing the new Dialog containing the response, or it can be done by simply calling and having it continuing with its logic. This can be seen done in Code Excerpt 1, where if the value of ServiceToCreate.QuotePrice equals zero, the Chatbot will request the user for more input and, if not, will continue the flow in the PrintService Dialog.

```
if (ServiceToCreate.QuotePrice == 0)
{
  await context.PostAsync("A data que pediu não está disponível. Para
  quando quer marcar?");
  context.Wait(ServiceDateAnswerAsync);
}
else
{
  await PrintService(context);
}
```

Code Excerpt 1 - Calling of new Dialog example

Another way a Dialog can call another one is with the PromptDialog. This is a particular type of Dialog that asks for something from the user, defined with the PromptOptions object. This object includes the query to make to the user, the specific choices of responses he can answer with, how many attempts the user can make before the Dialog realizes a problem might have occurred and what to tell the user when this situation happens. The object requires the ResumeAfter parameter, which is the Dialog that will be called after the user's answer.

A use of a PromptDialog can be seen in Code Excerpt 2. In it the Chatbot will ask the user a question, give it the possible options of answers and then it will call the LoginAnswerAsync Dialog with the user's response. If the user attempts to respond differently from the choices given by the Chatbot, the Dialog will throw a TooManyAttemptsException that can be caught inside the LoginAnswerAsync Dialog, being then able to handle it as the developer wants to.

```
var promptOptions = new PromptOptions<string>(
  prompt: "Precisa de estar autenticado para ver os seus serviços, quer
  ligar-se à sua conta Wegho?",
  retry: "Desculpe, não percebi. Utilize as opções apresentadas.",
  options: new List<string>() { "Sim", "Não", "Não tenho conta" },
  attempts: 1
);
PromptDialog.Choice(
  context: context,
  resume: LoginAnswerAsync,
  promptOptions: promptOptions
);
```

Code Excerpt 2 - PromptDialog example

6.4.2 State Data

The Chatbot's conversations' state is saved using the Entity Framework on two tables in a SQL Database, the Logs and the StateData. The first persists all turns of the conversation, every message that is sent by or to the user. The StateData is where the actual state is persisted,

including the user's data on the conversation. The implementation of this can be seen designed in Figure 20.

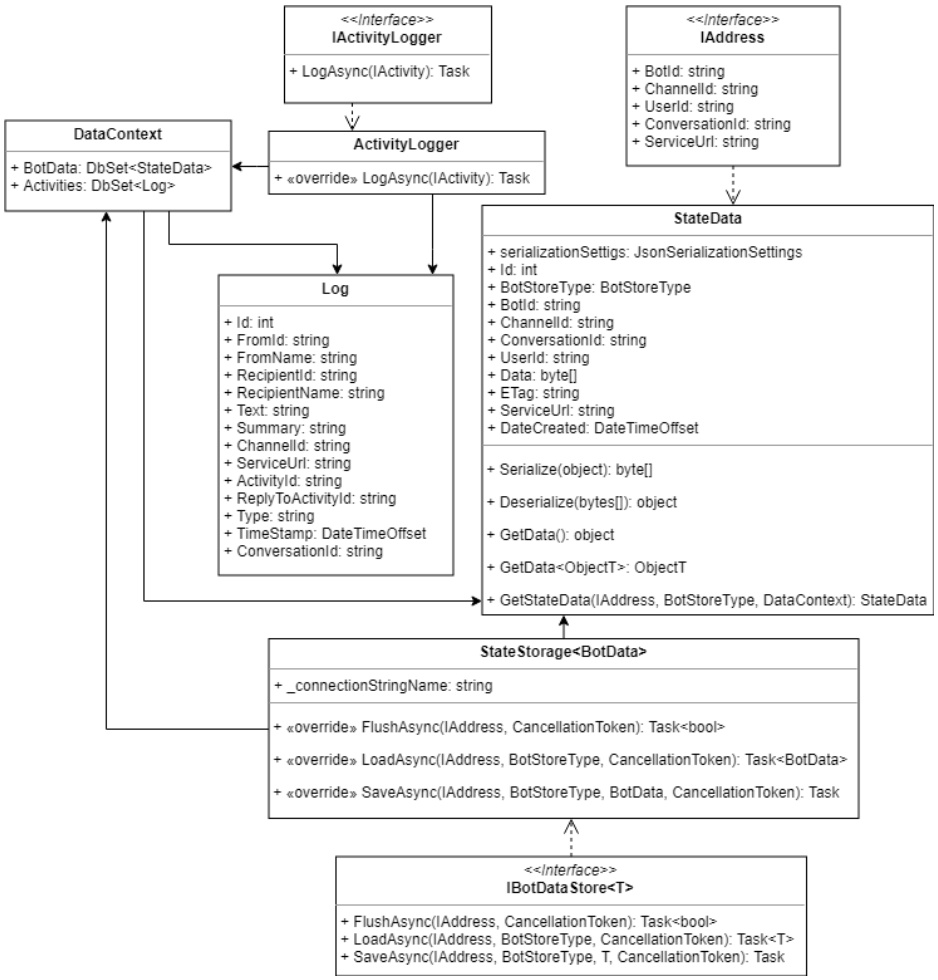

Figure 20 - Entity Framework Class Diagram

6.4.3 Web Services

For the connection between the Chatbot and the Wegho backend, a set of web services were developed by the team that also developed the Wegho platform, including the website and the mobile apps. These web services are called throughout the Chatbot code by HTTP REST web requests.

6.4.3.1 Get Token

This web service receives either the user's Facebook ID, e-mail address or phone number, and checks if there is an account in the Wegho database with the information given. If a user account

is found, it returns the user's e-mail address for future use, if it wasn't the detail given, and a unique token. This token has an expiration period of 30 minutes and will be required by the web service returns the user's bookings, as described in section 6.4.3.3.

6.4.3.2 Validate Service by Zip Code

This web service will be used to verify if a given service sub type is available for a specific location, identified by its Zip Code. It is used when a user attempts to create a new service, confirming if the requested location is available, or if a user directly asks for this information.

6.4.3.3 Bookings by User

This web service will use the token returned by the Get Token web service, as previously described in section 6.4.3.1, the user e-mail address and, if given, a date span, to return all the services previously scheduled by the user. This is used in the workflow explained in Section 5.3.4.

6.4.3.4 Get Service Information

This web service will be used any time a user's given details for a new service, as described in section 5.3.3, need to be validated. It receives the type of service to create and returns the following details:

- Minimum date available;
- Accepted service duration hours;
- Default service duration;
- Number of minutes per room (for domestic services);
- Accepted number of rooms;
- Number of minutes per bathroom (for domestic services);
- Accepted number of bathrooms.

These details are then all compared with the user's given information, in order to verify that a new service to be scheduled can be accepted within the Wegho service logic.

6.4.3.5 Create Quote

This web service takes all information that a user gives for a potential new service, either just by simulating it or fully asking to create one and returns the calculated price.

6.4.3.6 Get Frequencies by Service

This web service receives a service type and calculates the discount percentage that a frequency could have on a service to be scheduled. This discount will occur when a user requests a new service that would be repeated weekly, once a fortnight or monthly.

6.4.4 Zendesk Connection

Every time a user finishes a request that needs to be concluded by the Customer Support, a new Zendesk ticket needs to be created. This is created with the use of the Zendesk Core API (Zendesk, 2018), more specifically by creating requests. A request is a user's perspective on a ticket and is chosen in contrast to creating a regular ticket due to its ability to declare the requester's name. This ability will help the Customer Support later to handle the tickets, because if this was not declared, all tickets would go as belonging to the name of who sent the ticket creation request, which in this case would be the Zendesk Wegho manager.

Each ticket created will include all the information the Chatbot, at the time, has of the user and all the details required by the Customer Support, to efficiently complete the request.

6.4.5 Scorable Dialogs

As previously mentioned in Section 6.3.1 the user will have the possibility of pressing buttons on a persistent menu that will be available to them on FB Messenger. Since these buttons can be pressed at any given time, regardless of whether the user is in the middle of a Dialog, the Chatbot will use Scorable Dialogs to handle them.

Scorable Dialogs monitor all incoming messages and determines whether a message is suitable for an action (Microsoft Azure, 2017 c). When one is found, the respective Scorable Dialog is pushed to the top of the Dialog stack and handles the request accordingly.

As seen in the Class Diagram in Figure 7, to set the Scorable Dialogs up, the CommonResponsesScorable class was created, which will check if any message received by the Chatbot equals the keywords relating to the persistent menu buttons. If such a message is found, the Scorable class will call the CommonResponsesDialog, sending the keyword. This Dialog will then handle the user's request, depending on the keyword, and will return to the Dialog that was previously on top of the stack, if the one requested by the Scorable didn't clear it.

6.5 Supplier Logic App

To setup the Logic App that will handle the Supplier's check-in verification a new instance of this service was created on the Azure Portal. After creating the app, the construction of the workflow was made easily due to its visual designer tool. With it, the necessary connectors were

created and implemented, including a SQL connector to execute the query on the Wegho database, the Twilio connector that will send the message to the supplier. Another connector for each of these was implemented, which will be the connection to the actual SQL Server and the Twilio account, after authenticating, allowing for its use. Adding simple if/else logic to the workflow to verify if the query runs with no problem and returns any service with check-ins not made on time and the logic app was ready.

Another logic app was also developed for when a user responds to a SMS. This logic app will receive the user's response and through an Email connector will send it to the Customer Support agents, so that they can choose how to deal with it. This logic app's trigger will be an HTTP request, which will create an endpoint that will, afterwards, be setup as the Webhook on Twilio for when a message comes in on that specific number (Cruz, 2017).

6.6 Main Problems

Throughout the development of this project several problems occurred, most of them being solutioned, although some ended up not having an ideal solution, or even any.

6.6.1 Persistent Menu functioning without thread control

This problem occurred when a persistent menu button was pressed, and the thread control did not belong to the Chatbot. This would cause the Chatbot to respond to that action request, but only that first action. It would ask something of the user and when the user would respond, the Chatbot would not react. This happened because only the message event sent by the FB Messenger would end up on the Chatbot and all other messages were correctly ignored. This would cause misunderstanding to a user that would believe he was no longer having a conversation with the Chatbot, in spite of having it responding to some commands and, even then, not following the workflows of those actions. This issue was solutioned as is described in Section 6.3.4.

6.6.2 New messages not appearing in Facebook Messenger

This issue was found when comparing the logs of the Chatbot and the conversations with the users, after the testing process was started. What was shown was that some conversations

between the Chatbot and a user were not appearing on the Facebook Page, only on the logs, and therefore there was no way of having the Customer Support supervising them, other than giving them access to the database or creating a whole new tool where the conversations could be scrutinized.

Considering this issue, reviewing the logs and the states of the conversations and how they were initiated, on the Facebook page or with the use of the Chat Plugin on the Wegho website, no cause for the problem was found. After testing with new users, on multiple platforms and using different approaches, no differences between the conversations that appear to the Customer Support and those that don't were discovered.

The only way these conversations seemed to appear was after the user would request to be redirected to the Customer Support and after they sent any message next to the request. Due to this, a simple message is always sent to the user, after the redirection asks if he wants to give any more details. This will give the user the reaction to respond, making the whole conversation appear for the Customer Support to see, even the messages sent before the redirection.

A support ticket was made to the Facebook team to try and find what the cause of this situation was, but till the writing of this dissertation writing no answer was received yet.

6.6.3 Customer Support requesting thread control not working

The Customer Support can always supervise the conversations between the Chatbot and the user (except if the problem described in Section 6.6.2 occurs), and if they decide they can interject the Chatbot with input of their own. The best way to do this is to take thread control away from the Chatbot and interact directly with the user by pressing the "Move to Inbox" button on the FB Messenger page.

At first, this was not working because pressing of that button did not take thread control, but only requested it. Therefore, the solution described in Section 6.3.4 was designed and implemented and this problem was solved.

6.6.4 Connection between Chatbot and pre-existing Wegho web services

The way the connection between the Chatbot and the Wegho back-end was supposed to work was to give it full access to the web services that would be required to create and manage

services, so that no interaction from the Customer Support would be essential for these issues. This ended up not being possible to be done due to the design of the already existing web services that needed these to be made inside the website or the mobile apps, using their already built in authentication.

This was not able to be replicated with the Chatbot, and so the solution of having Zendesk tickets being created for the Customer Support to handle the requests manually was chosen, as described in Section 6.4.4.

6.6.5 Adaptive Card not working on Facebook Messenger

When at first developing this project, the idea was for when a service was to be shown to the user, it would be presented to him by the use of an Adaptive Card (Microsoft, 2017). These cards were defined by a JSON object, having multiple templates designed, one for each service type. These would attractively show the user all the information that was saved about a given service, either one that was being created or one already existing. These cards would also have buttons attached, with the actions that the user would be able to take. An example of such a card can be seen in Figure 21.

Figure 21 - Example of functioning Adaptive Card

Although this was first functioning as expected, the adaptive cards suddenly stopped properly working, rendering all the information to an image that was zoomed in and couldn't be opened to be fully seen, as shown in Figure 22.

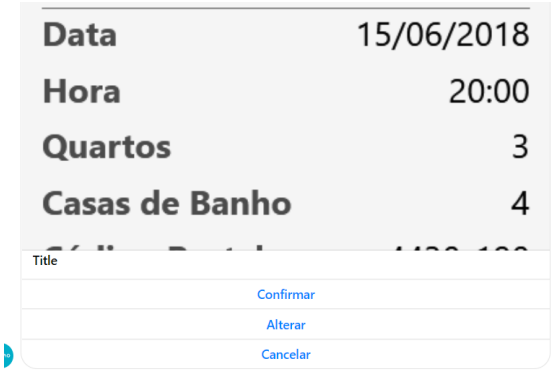

Figure 22 - Adaptive Card not working on Facebook Messenger

Assistance on this issue was requested to both the Microsoft team that developed the Adaptive cards (GitHub, 2018 a) and the FB Messenger team, in order to try and figure out what the problem was and if there was a solution to it, but no such solution was found.

What was decided needed to be done was that the Chatbot would simply print the information to the user, giving him prompt choices as described in Section 6.4.1.

6.6.6 QnA Maker Dialog inside other Dialogs

The first way the integration with the QnA Maker was developed was through the use of the QnAMakerDialog (GitHub, 2018 c), a tool that allows easy setup of these. This integration would work by having the Bot Builder call this Dialog any time an Intent was not found by LUIS, in order to see if the user sent message was a question that was in the QnA Maker knowledge base.

The problem with this solution was that the Chatbot would only answer the user's questions if the user wasn't already inside another workflow.

Thus, another implementation was developed through the creating of the nested class QnAMakerAnswer, as described in Section 5.2.2. This class will be used during the runtime of the Chatbot, any time the user sends a message. This message will then be sent to the QnA Maker API and if it is found to be a question in its knowledge base, its corresponding answer

will be sent back to the user, momentarily interrupting the current flow of the conversation. If no similar question is to be found in the knowledge base then the Chatbot will reply that it didn't understand, or if it is in the middle of a workflow it will assume to be in response to the last message sent by the Chatbot.

7 Experimentation and Evaluation

This section contains the description of the methods on which the developed solution will be tested and evaluated, so as to have a perception of the success of it.

7.1 Metrics

After reading the study made by Chakrabarti and Luger on the architecture, algorithms and evaluation metrics for customer service chatbots (Chakrabarti & Luger, 2015), it was established that evaluating the utterances using Grice's cooperative maxims is an effective way to compare chatter bots. This method provides a scoring matrix, against which each artificial conversation can be graded for a specific criteria. These four maxims are the following:

1. Quality: speaker's utterance is as the truth as provable by adequate contextual evidence or domain facts;
2. Quantity: speaker's utterance provides as much information as appropriate, not more, nor less;
3. Relation: speaker's utterance is relevant to the context and the topic of the conversation;
4. Manner: speaker's utterance is direct and straightforward, without ambiguity or obfuscation.

Another three metrics, hypothesized by Chakrabarti and Luger as being helpful to evaluating chatbots, are the ones based on coherence and solving problems.

5. The first is the percentage of successful resolutions. It is straightforward, simply calculating all the conversations made with the purpose of solving customer issues, which leads to the percentage of all that ended with success, without the assistance of a human agent.
6. Another metric to evaluate is the percentage of follow-up questions. For this project, the conversation for the creation of a new service will be tested, acknowledging that a specific set of variables are required so that a scheduling can be made. The chatbot asks all the variables needed that haven't already been provided by the customer and, by the end of the conversation, there will be a score retrieved from the percentage of questions required to be asked.

7. Finally, the last metric to be used will be the number of coherent turns a conversation has had. This value will be studied by reading the test conversation transcripts and counting the number of utterance exchange pairs that were made before an incoherent answer or a lack of understanding statement is given by the chatbot.

Out of the seven metrics, four of them, Grice's quality maxim, percentage of follow-up questions, percentage of successful resolutions and number of coherent conversation turns, are evaluated by examining the conversation transcript, or by looking up at the response from the domain knowledge. The other three metrics, Grice's quantity, relation, and manner maxims require subjective evaluations.

At the end of each solving conversation issue, the customer will be asked if he approves the functioning of the Chatbot, through a simple "yes" or "no" question. He will also be asked for free comments, either positive or negative ones. This will give another metric, the user's satisfaction, and will enhance the opportunity to receive feedback on how to improve the functioning of the service.

7.2 Hypothesis

The hypothesis to be tested so as to support the results of this project is that a well-defined and functioning Chatbot can be a powerful aide to Customer Service, capable of handling simple client issues and therefore freeing much needed time for its human agents to handle more complex issues.

7.3 Evaluation Methodology

For the evaluation of the Chatbot, testing groups were used, having them have full conversations with the service with different issues to be solved without the assistance from a human agent.

These issues involved creating a new service, modifying or cancelling an existing one and asking questions about the functioning of the service. The users were also encouraged to go outside the given tasks and improvise, so as to evaluate how the Chatbot could handle the problems encountered.

The testing groups defined were Internal, External-Closed and External-Open. The first group that tested the Chatbot were the Wegho company's employees, mainly those that had already worked on the Customer Support service within the platform. These users exclusively tested the Chatbot for one week after it was published.

The second group, External-Closed, were testers that were directly invited to test the Chatbot. These users, mostly friends and colleagues of the Wegho team members, when invited received some instructions on how to best work with the Chatbot. They started testing it after the first week of the Internal group, and both had one more week of exclusive access to the Chatbot.

After these first two weeks, the Chatbot was published and set as public, being available to start dealing with any questions a new user might make on the Wegho Facebook page. The External-Open group was made out of these users and they tested the Chatbot for 6 weeks, before the results were extracted for the writing of this dissertation.

At the end of each task being completed the user was asked for any feedback, as well as his/ her satisfaction, so that all the required information by the metrics could be extracted. This task was removed after the testing period, leaving the Chatbot published and public though not saving any more testing results.

7.3.1 Tests Database

The tests results populated a SQL Database on an Azure SQL Server which can be observed designed in the Figure 23. This was filled using Logic Apps, as explained in Section 7.3.2.

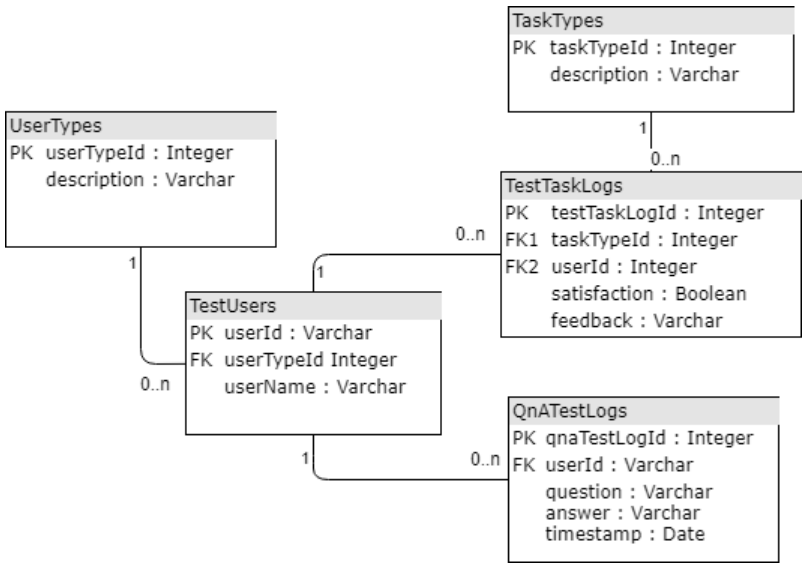

Figure 23 - Tests Database

This database consists of the following five tables:

- UserTypes: This table was fixed when implemented and includes the three different test groups explained in Section 7.3;
- TestUsers: Here information about the user that talks with the Chatbot is persisted, having them be assigned one UserType from the previously described table;
- TaskTypes: This table was also fixed upon its implementation, and it includes every different task a user may try with the Chatbot. These are:
    - Authenticate;
    - Create Domestic Service;
    - Create Office Service;
    - List Services;
    - Modify Service.
- TestTaskLogs: Here is where after a user accomplishes a task, it will be persisted his satisfaction and feedback, if given. Each element in this table is associated to one TestUser element and one TaskType element;

- QnATestLogs: Here is where, after any call to the QnAMaker, the question and answer returned is saved and associated to the TestUser that made it.

The Logs table of the Chatbot, described in Section 6.4.2, was also inspected for the test results.

7.3.2 Logic Apps

To populate the tables previously described, multiple Logic Apps were created, one for creating users, one for the test task logs and one for the QnA test logs. These logic apps were designed with HTTP endpoint triggers to be called by the Bot Builder logic at any time needed. Each will receive the necessary information by the table and will insert a new line, except for the task of creating new users, which will first check if the user doesn't exist yet.

7.4 Results

The metrics described in Section 7.1 were divided in two sub-groups when receiving their results, having the first four being tested from the QnA Test logs, and the other three from the Task tests.

7.4.1 QnA Tests Results

The first thing revealed after examining the QnA test logs was that there was a significantly large number of questions of which answers were not to be found in the QnA Maker knowledge base. As can be seen in Table 7, over 87% of all questions made to the Chatbot ended having no answer known to it. While these logs are useful as they can be used to improve the knowledge base moving forward, for these tests, it lowers the inherent value that can be taken from these results. This is because the scores of the first four metrics described in Section 7.1 only make sense if taken from the questions that have answers, showing that for every other question the score for each would instantly be zero.

Table 7 - Percentage of questions of with no answer found

	External-Closed	**External-Open**	**Total**
Percentage	88.89%	87.59%	87.72%

Nevertheless, these metrics were studied and the first was the Quality score, seen in Table 8. Since the answers given by the Chatbot were manually inputted into the QnA Maker knowledge base and had on its basis the FAQ section of the Wegho website and the general function of the platform, this score is the maximum for every test group. This score should always be kept at its maximum since at no other point should the Chatbot give a response that isn't the truth, even if that response has nothing to do with the question asked.

Table 8 - Quality Scores

	External-Closed	**External-Open**	**Total**
Average	1.000	1.000	1.000

Next is the Quantity score, which, as seen in the Table 9, in its total is 0.410, below half of its potential maximum score. After verifying the logs, it was verified that some of the answers given by the Chatbot provide more information than requested by the user. For example, one might ask the Chatbot if a domestic service includes the cleaning of the kitchen, to which it would respond by telling the user everything that's included in the service. While the user's question might be answered, he ended up receiving more than what was explicitly asked. The way to improve this score would be to divide those questions that give more information, into smaller answers. In the example given, while the existing question/answer pair should be kept for when a user asked for what a domestic service included, different pairs for each detail could be added to the knowledge base, so that when a user specifically asked for some information, he would receive a more specific answer.

Table 9 - Quantity Scores

	External-Closed	**External-Open**	**Total**
Average	0.667	0.365	0.410

The next metric is the Relation score, and this is the one that could potentially be improved by the more tests done and the more improved the knowledge base is. What this score basically indicates is if the answer given by the Chatbot is the correct one for what the user questioned. As seen in the Table 10, the final score for all the users that tested the Chatbot was half of its maximum score, which means that half of the answers given were the correct one, or, at the very least, were related to the questions asked.

Since the variety of ways a user can make a request or ask a question is very big and, given that the complexity of the Wegho platform is also huge, providing many kinds of services, this score can hardly reach its maximum value. But the better trained the knowledge base is and the bigger the quantity of question/answer pairs it has, the higher this score can get, the better to maintain the Chatbot at its best, always working to improve it.

Table 10 - Relation Scores

	External-Closed	External-Open	Total
Average	0.667	0.471	0.500

And the last of the four metrics is the Manner score. Like the Quality score, this, as seen in Table 11, has its maximum score and, as already said, this should always be kept at its maximum. The Chatbot should never give an answer that isn't direct and should always maintain its lack of ambiguity. Even if the answer provided by the Chatbot isn't exactly what the user requested, it should give its information as clear as possible.

Table 11 - Manner Scores

	External-Closed	External-Open	Total
Average	1.000	1.000	1.000

7.4.2 Task Tests Results

When studying the results of the tests, what can be seen first is that there are few result logs for the Task tests. As seen in Table 12, of all the users that had conversations with the Chatbot, only 8.33% of these concluded any workflow. Even though this value increases to a 35.71%, just over a third when looking at users of the External-Closed test group, this limits the overall assumptions that can be taken from these results.

This should have been accounted for, and a solution would be to calculate metrics for all the users that started any workflows instead of only those that concluded them.

Table 12 - Percentage of Users that Finished Tasks

	External-Closed	External-Open	Total
Percentage	35.71%	6.15%	8.33%

When looking at the percentage of successful resolutions in Table 13, the fifth metric described in Section 7.1 what is shown is the big difference between the value of each test groups. 52.17% of the tasks made by the External-Closed group users were concluded with successful resolutions, in contrast to only 13.33% of the External-Open group users. This showed the difference that was made when receiving any instructions or help on how the Chatbot operated. By looking at the logs of the conversations, another setback was that since the users in the External-Group were using the Chatbot for the actual purpose of testing it, they had a higher predisposition to finishing their tasks. This, unlikely the users of the other group that were already using the Chatbot for the actual practical use of creating or managing a service, was a setback. As they could give up on the Chatbot and end it up, either by using the platform in its other available forms (website or mobile app) or by requesting to be redirected to the Customer Support to directly so as to ask a human agent for help.

Table 13 - Percentage of Successful Resolutions

	External-Closed	**External-Open**	**Total**
Percentage	52.17%	13.33%	36.84%

The next metric studied was the percentage of follow-up questions, the sixth described in Section 7.1, and here what can be first seen in the Table 14 is the similarity of the values between the two groups, with only a 7.03% percentage difference between the two. With this small difference, what can be assumed is that even though the users from the External-Closed group had instructions on how to request to create a service and may give some more details of what exactly they want, they will not proactively give much more than the users in the External-Open group.

Another thing that could be inferred from these values, specifically the total of 80.19%, was that users needed the Chatbot to ask them the details of the services to be created. Only 19.71% percent of the information required did not have to be directly asked and, after looking at the conversation logs, what could be confirmed was that most of these information were calculated by the Chatbot. For example, a domestic service has a given duration set for it and the user can directly request another specific duration. If he doesn't, then the Chatbot will automatically calculate its duration be from the number of rooms and bathrooms given by the user.

Table 14 - Percentage of Follow-Up Questions

	External-Closed	**External-Open**	**Total**
Percentage	77.97%	85.71%	80.19%

The next metric studied was the number of coherent turns before the Chatbot would give an incoherent response to a user utterance. The average for this number can be seen in Table 15. From these results it was shown that, disregarding from which test group it was, the user having a conversation with the Chatbot took on average four or more exchange pairs in order for the Chatbot to be incoherent after they started a new task request.

But these values must take the fact into account that, as can be seen in Table 16, just less than 40% of the users from the External-Closed group had incoherent turns, while this number was by far higher for the users of the External-Open group. Here, almost 90% of all users had at one time or another an incoherent answer from the Chatbot. Hence, the users that had no prior instruction on what to do, or even what the Chatbot could exclusively do, had a bigger chance of, at some point during the conversation, receiving an incoherent answer or having the Chatbot not understanding what they had requested.

Table 15 - Average of Coherent Turns

	External-Closed	External-Open	Total
Average	4.56	4.23	4.36

Table 16 - Percentage of Existing Conversations with Incoherent Turns

	External-Closed	**External-Open**	**Total**
Percentage	39.13%	86.67%	57.889%

8 Conclusions

In this section, the final remarks on the dissertation will be made, emphasizing on which objectives were completely accomplished and which were not, what limitations the final product has and what could be improved from now on, as well as the overall balance of the project.

8.1 Accomplished objectives

In terms of the objectives described in Section 1.3 most were accomplished, even though some ended up not being developed as originally envisioned.

1. Creation of a new service:

This objective was accomplished, in spite of the fact that the service created couldn't be automatically inserted in the Wegho system and had to be manually inserted by the Customer Support.

2. Modification of an existing service:

This objective was accomplished, even though the service modified couldn't automatically be updated in the Wegho system and had to be manually done by the Customer Support.

3. Cancellation of an existing service:

This objective was also accomplished, the only exception being that instead of the service cancelled being automatically cancelled into the Wegho system it had to be manually done by the Customer Support.

4. Assisting supplier when no check-in has been made:

This objective was accomplished, even though it ended up not being done as originally intended and being now handled by an Azure Logic App instead of the Chatbot.

5. Understanding of the user:

This objective was partially completed since the Chatbot mostly understands the user's requests but needs to be improved from now on.

6. Direction of user to customer support when it fails to understand its intent:

This objective was fully accomplished.

7. Direction of user to customer support when directly requested:

This objective was fully accomplished.

8. Availability of Chatbot in multiple channels:

This objective was not accomplished due to decisions made during the design portion of the Chatbot, ending up only being available on the FB Messenger channel.

8.2 Limitations and further improvements

The main limitation that the Chatbot developed has is that it will never be completely concluded. Due to the complexity of the human language, there will always be room to the improvement of its understanding skills. This constant improvement is something that should be done throughout the Chatbot's lifespan, always enhancing its ability to understand and what information better suits the users through the updating of its QnA Maker knowledge base and the utterances associated to its LUIS intents and entities.

Also the problem that was described in the Section 6.6.4, i.e. the inability to automatically create, modify or cancel services, requiring the final manual action from the Customer Support, doesn't free as much work out of those agents' hands as it potentially could or was expected to. Hence, if in the future this issue could be solved and implemented, it would certainly be a big improvement into the final product.

The last improvement that could be made to the Chatbot is the increasing of the different types of services that could be created with it. As it stands, it can only schedule domestic cleaning and office cleaning services, all the other types being relegated to a general "Other" type. Since the Wegho platform includes many more different kinds of services, the Chatbot could also be prepared to create and implement them.

8.3 Final appreciation

The Chatbot created with this project, although not being as independent as was first envisioned, still managed to fulfil most of its objectives. And even though it has a great potential of improvement, both on its language understanding and on its functionalities, it still ends up freeing much of the Customer Support's time. Due to this program, every user that speaks with the Wegho's Messenger platform, either using the Chat Plugin in the website or the Facebook Page, first talks with the Chatbot. Only after specifically asking for redirection, will have Customer Support have to interact with the user, diminishing the number of times they must or have to do it.

Having the Chatbot always available to all users will also help the Customer Support, as by using the Chatbot if a user requests help outside the service's serving working hours, he will not have to wait for the following work day, or even longer, to get a response. Nor will the user have to wait for his or her turn to being answered, given the Chatbot's ability to answer as many users as needed at any given time.

Overall the final product developed with this project, in spite of having some issues to be dealt with, was a positive addition to the Wegho platform, and, if correctly managed and improved, may well become more and more prepared to correspond to the users' issues. Since most of its problems are based on the lack of understanding due to the necessity of more utterances, through the process of improving both the LUIS Intents and the QnA knowledge base, these problems could and should become less and less frequent. This would potentially diminish the number of times the Customer Support would have to directly interact with the users, leading to freeing valuable time to solving other important issues, which was one of the goals intended to be achieved.

9 References

Arcand, K., 2017. Can Chatbots Fully Replace Humans? Not Yet. *Customer Relationship Manager,* p. 4.

Chakrabarti, C. & Luger, G. F., 2015. Artificial conversations for customer service chatter bots: Architecture, algorithms, and evaluation metrics. *Expert Systems with Applications 42,* pp. 6878-6897.

Chatfuel, 2018. *Chatfuel.* [Online]
Available at: https://chatfuel.com/

Cruz, P. d. l., 2017. *Triggering an Azure Logic App by SMS messages with Twilio.* [Online]
Available at: https://blog.mexia.com.au/triggering-an-azure-logic-app-with-an-sms-using-twilio

Facebook, 2018 a. *Graph API User.* [Online]
Available at: https://developers.facebook.com/docs/graph-api/reference/v2.2/user

Facebook, 2018 b. *Handover Protocol.* [Online]
Available at: https://developers.facebook.com/docs/messenger-platform/handover-protocol/

Facebook, 2018 c. *Handover Protocol API Reference.* [Online]
Available at: https://developers.facebook.com/docs/messenger-platform/reference/handover-protocol

Facebook, 2018 d. *persistent_menu Reference.* [Online]
Available at: https://developers.facebook.com/docs/messenger-platform/reference/messenger-profile-api/persistent-menu/

Facebook, 2018 e. *The Welcome Screen.* [Online]
Available at: https://developers.facebook.com/docs/messenger-platform/discovery/welcome-screen/

Facebook, 2018 f. *Customer Chat Plugin (beta).* [Online]
Available at: https://developers.facebook.com/docs/messenger-platform/discovery/customer-chat-plugin/

Fluss, D., 2017. The AI Revolution in Customer Service. *Customer Relationship Manager,* p. 38.

FRPT Research, 2017. Artificial Intelligence powered chatbots failing due to disappointing. *FRPT - Software Industry Snapshot,* 1 June.pp. 3-4.

Fryer, L. K. et al., 2017. Stimulating and sustaining interest in a language course: An experimental comparison of Chatbot and Human task partners. *Computers in Human Behavior 75,* pp. 461-468.

GitHub, 2018 a. *Adaptive Card on Facebook.* [Online]
Available at: https://github.com/MicrosoftDocs/AdaptiveCards/issues/73

GitHub, 2018 b. *Facebook Messenger Handover Protocol with Handle Activity Type Middleware.* [Online]
Available at: https://github.com/BotBuilderCommunity/botbuilder-community-dotnet/issues/7

GitHub, 2018 c. *QnAMakerDialog.* [Online]
Available at: https://github.com/garypretty/botframework/tree/master/QnAMakerDialog

Hill, J., Ford, R. W. & Farreras, I. G., 2015. Real conversations with artificial intelligence: A comparison between human–human online conversations and human–chatbot conversations. *Computers in Human Behavior.*

IBM, 2018 a. *Conversation - API Reference.* [Online]
Available at: https://www.ibm.com/watson/developercloud/conversation/api/v1/

IBM, 2018 b. *Conversation: About.* [Online]
Available at: https://console.bluemix.net/docs/services/conversation/index.html#about

Ishida, Y. & Chiba, R., 2017. Freel Will and Turing Test with Multiple Agents. *Procedia Computer Science 112,* pp. 2506-2518.

Knight, W., 2016. A Plague of Dumb Chatbots. *MIT Technology Review,* p. 28.

Koen, P. A., Bertels, H. M. J. & Kleinschmidt, E. J., 2014. Managing the Front End of Innovation - Part II. *Research-Technology Management,* pp. 25-35.

Korzeniowski, P., 2017. Bots Should Be In Your Contact Center's Future. *Customer Relationship Manager,* pp. 28-32.

Microsoft Azure, 2017 a. *About Bot Service.* [Online]
Available at: https://docs.microsoft.com/en-us/bot-framework/bot-service-overview-introduction

Microsoft Azure, 2017 b. *About Language Understanding (LUIS).* [Online]
Available at: https://docs.microsoft.com/en-us/azure/cognitive-services/LUIS/Home

Microsoft Azure, 2017 c. *Global message handlers using scorables.* [Online]
Available at: https://docs.microsoft.com/en-us/azure/bot-service/dotnet/bot-builder-dotnet-scorable-dialogs?view=azure-bot-service-3.0

Microsoft Azure, 2018 d. *Connect a bot to Facebook Messenger.* [Online]
Available at: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-channel-connect-facebook?view=azure-bot-service-4.0

Microsoft Azure, 2018 e. *Dialogs Library.* [Online]
Available at: https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-concept-dialog?view=azure-bot-service-4.0

Microsoft Azure, 2018 f. *LUIS: Homepage.* [Online]
Available at: https://www.luis.ai/home

Microsoft Azure, 2018 g. *QnA Maker.* [Online]
Available at: https://azure.microsoft.com/en-us/services/cognitive-services/qna-maker/

Microsoft, 2017. *Adaptive Cards Overview.* [Online]
Available at: https://docs.microsoft.com/en-us/adaptive-cards/

Peeples, D., 2016. The Rise of Chatbots: Proestream Brings Secure Mobile Messaging to Insurance Industry. *Insurance Advocate,* pp. 20-22.

Sagarin, R., 2014. Customer Service Needs to Be Either More or Less Robotic. *Harvard Business Review,* pp. 2-5.

Saran, C., 2017. Chatbots lead the debate on public sector use of AI to streamline digital services. *Computer Weekly,* pp. 4-6.

Skerret, D., 2017. Seven Reasons Why Most Chatbots Launched in 2017 Are Dead on Arrival. *EContentMag,* p. 15.

Waxer, C., 2016. Get Ready for the Bot Revolution. *Computerworld Digital Magazine,* pp. 8-16.

Zendesk, 2018. *Core API - Requests.* [Online]
Available at: https://developer.zendesk.com/rest_api/docs/core/requests

10 Annexes

10.1 Bot Builder Class Diagram

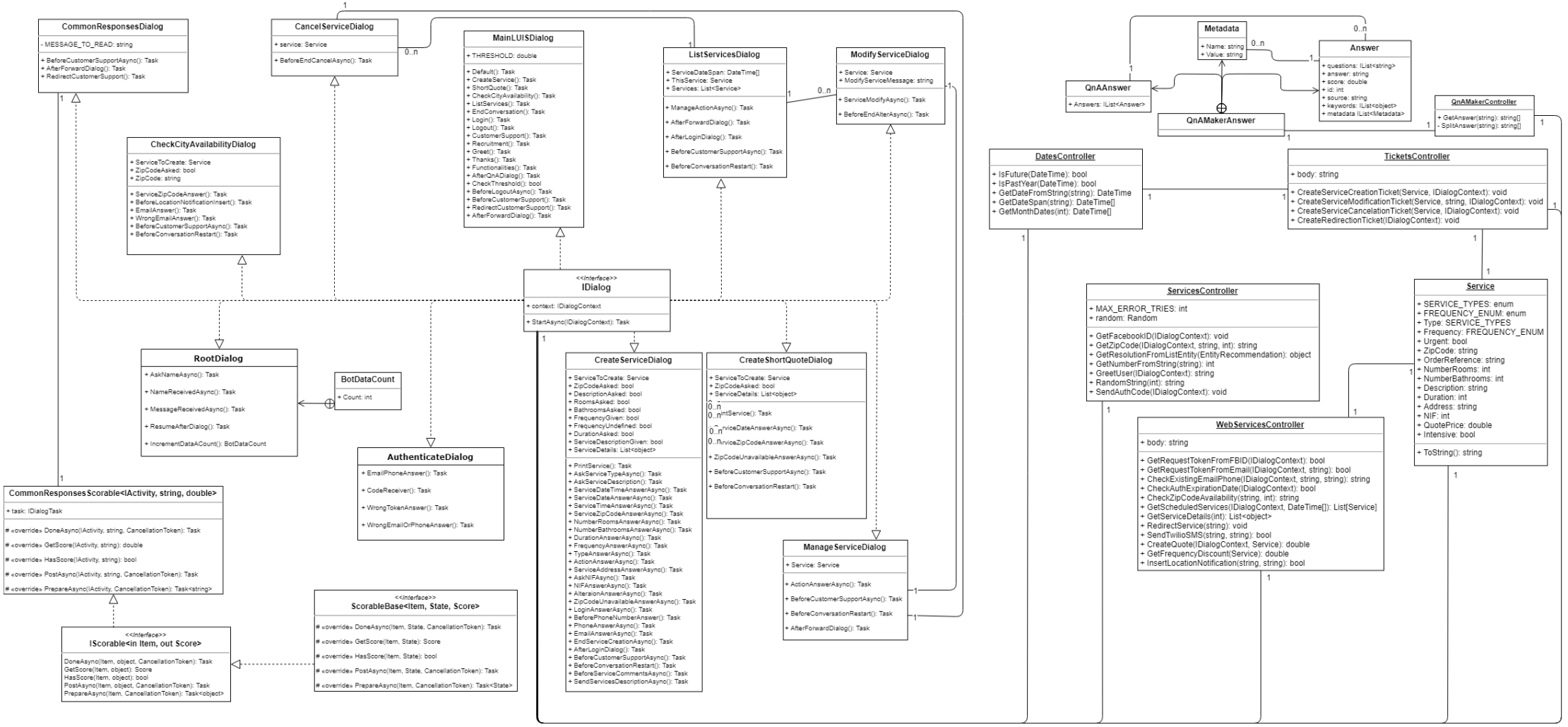

Figure 24 - Bot Builder Solution Class Diagram