UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

Solução Mobile – Voice Assisted Chatbots

João Duarte Gonçalves Palma

Mestrado em Engenharia Informática
Especialização em Engenharia de Software

Versão Pública

Trabalho de projeto orientado por:
Professor Doutor Rui André Oliveira

2019

Agradecimentos

Antes de mais, agradeço aos meus pais pelo amor incondicional e por sempre me terem proporcionado as condições ideais ao longo de todo o percurso académico. Sem o vosso apoio não seria possível ter chegado onde cheguei.

Ao meu irmão por ter estado desde sempre do meu lado, independentemente da situação, e pela pessoa admirável em que se tornou.

Aos meus avós e ao meu tio pelo seu apoio e por acreditarem sempre em mim.

Aos meus amigos e colegas de trabalho, Pedro e Vítor, por terem sido bons companheiros ao longo deste ano. Pelo ambiente de boa disposição que proporcionaram e por estarem sempre disponíveis para me ajudar.

Aos meus amigos de longa data, Verónica e Diogo, pelos muitos conselhos e pelo tempo que despenderam por causa desta tese. Obrigado pelo vosso apoio incondicional.

Aos meus amigos e colegas de Faculdade, principalmente os do meu ano, por me terem acompanhado ao longo de toda a vida académica e por continuarem a ser um pilar fundamental na minha vida.

Ao pessoal dos JSF Lisboa, principalmente ao Cabrita, à Ana e à Inês, pelo seu companheirismo e constante boa disposição ao longo destes meses.

Ao meu orientador da Faculdade, Professor Doutor Rui Oliveira, pelo seu suporte eficaz nas alturas de dificuldade e por ter acreditado nas minhas capacidades desde o início.

À equipa de FFM, especialmente ao meu orientador, Sérgio Davide Gaio, pela forma espetacular como me acolheu na *Accenture* e por todo o seu apoio.

Por fim, e acima de tudo, a Deus.

Resumo

Face à crescente evolução da área da mobilidade e ao aumento significativo do número de dispositivos móveis em utilização nos últimos anos, faz cada vez mais sentido apostar no desenvolvimento de soluções móveis que permitam facilitar certas tarefas do dia-a-dia. Adicionalmente, o uso de *chatbots* nas mais diversas áreas da indústria tem vindo a evidenciar as capacidades de otimização que os mesmos possibilitam.

Este relatório descreve o trabalho realizado no estágio feito na empresa *Accenture*, que teve como objetivo o desenvolvimento de uma solução móvel que funcionasse como assistente virtual, e que suportasse comunicação através de voz. O *chatbot* resultante deste projeto é capaz de responder a questões sobre problemas comuns em *boxes* de televisão, esclarecer o utilizador sobre questões relacionadas com faturas e carregamentos, e permitir o agendamento de assistência técnica, caso necessário. O projeto assentou no desenvolvimento de um protótipo que possa posteriormente ser apresentado como proposta de solução a clientes da *Accenture* da indústria das telecomunicações.

Numa fase inicial, foi feito um enquadramento do tema a partir de um estudo comparativo e aprofundado da literatura relevante. Este enquadramento foi sucedido por uma fase de planeamento e de análise de requisitos, que permitiu clarificar o âmbito do projeto e estabelecer uma base de trabalho concreta.

A fase seguinte consistiu no desenho da arquitetura do *chatbot*, tendo em conta os diferentes componentes a integrar e a forma como estes interagem entre si. Foram também considerados os aspetos de segurança relevantes para o desenho do sistema.

A terceira fase consistiu na implementação da solução proposta. Para isto, recorreu-se a vários tipos de tecnologias de forma a conseguir atingir as funcionalidades necessárias ao cumprimento dos requisitos funcionais. O *chatbot* criado, inicialmente disponível apenas localmente, foi hospedado na *cloud* e integrado com uma base de dados, uma interface gráfica e com suporte de voz.

Por fim, foram efetuados testes com o objetivo de verificar e validar as características do *chatbot* desenvolvido e de obter *feedback* relevante para iterações futuras.

Palavras-chave: Chatbot, Computação Móvel, Assistente Virtual, Apoio ao cliente, Software

Abstract

In the face of the growing evolution in the area of mobility and the significant increase in the number of mobile devices in use in recent years, it's increasingly important to focus on the development of mobile solutions that facilitate certain day-to-day tasks. In addition, the use of chatbots in the most diverse areas of the industry has been evidencing the optimization capabilities that they enable.

This report describes the work during the internship at *Accenture*, which aimed to develop a mobile solution that works as a virtual assistant which supports voice communication. The chatbot created for this project is able to answer questions about common issues with television boxes, clarify the user on issues related to invoices and payments, and allow the scheduling of technical assistance, if necessary. The project was based on the development of a prototype that could later be presented as a solution proposal to *Accenture*'s customers in the telecommunications' industry.

During an initial phase, and based on a comparative and in depth study of the relevant literature, a contextualization of the problem was made. This contextualization was followed by a requirements planning and analysis phase, which made it possible to clarify the scope of the project and to establish a concrete work base.

The next phase consisted in the design of the chatbot architecture, taking into account the different components to integrate and the way they communicate with each other. The security aspects relevant to the design of the system were also discussed.

The third phase was the implementation of the proposed solution. To this end, various types of technology were used in order to achieve the functionalities necessary to fulfill the functional requirements. The created chatbot, initially only available locally, was hosted in the cloud and integrated with a database, a graphical interface and with voice support.

Finally, tests were carried out to verify and validate the characteristics of the developed chatbot and to obtain relevant feedback for future iterations.

Keywords: Chatbot, Mobile Computing, Voice Assistant, Customer Support, Software

Conteúdo

Acrónimos

ACID	Atomicity, Consistency, Isolation, Durability
AIML	Artificial Intelligence Modelling Language
API	Application Programming Interface
ASR	Automatic Speech Recognition
B2C	Business to Consumer
CDN	Content Delivery Network
DNN	Deep Neural Network
FCUL	Faculdade de Ciências da Universidade de Lisboa
FFM	Field Force Management
HMM	Hidden Markov Model
MVP	Minimum Viable Product
NLP	Natural Language Processing
NPM	Node Packege Manager
PEI	Projeto de Engenharia Informática
REST	Representational State Transfer
SDK	Software Development Kit
SDLC	Software Development Life Cycle
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
URL	Uniform Resource Locator
XML	Extensible Markup Language
XSS	Cross-site Scripting

Lista de figuras

Lista de tabelas

Capítulo 1

Introdução

O presente trabalho é realizado no contexto da disciplina de Projeto de Engenharia Informática (PEI) pertencente ao Mestrado em Engenharia Informática da Faculdade de Ciências da Universidade de Lisboa (FCUL). O projeto apresentado está inserido na área do desenvolvimento móvel e foi realizado em contexto empresarial, num estágio na empresa *Accenture*. Este primeiro capítulo apresenta uma contextualização do projeto, a descrição da instituição de acolhimento, os motivos e objetivos associados e a organização do documento.

A área da computação móvel tem mantido um crescimento significativo ao longo dos últimos anos, e ocupa cada vez mais uma posição de relevância no contexto do desenvolvimento de *software*. A crescente utilização de dispositivos móveis como *smartphones*, *tablets* ou *wearables*, que garantem aos utilizadores inúmeras possibilidades, é notória nos dias de hoje. Um estudo de 2016 realizado pela *StatCounter* mostra que o número de dispositivos móveis a aceder à *Internet* já é superior ao número de *desktops* [69], evidenciando ainda mais as oportunidades presentes na área da mobilidade.

Um *chatbot* é um agente de conversação que interage com humanos, turno a turno, através de linguagem natural. A primeira concetualização é atribuída a *Alan Turing*, que num artigo publicado em 1950 propôs um teste que media a capacidade de um computador personalizar um humano [70]. Desde *Turing*, as tecnologias por trás da construção dos *chatbots* têm evoluído a nível de aprendizagem automática e de processamento de linguagem natural. Da mesma forma, a adoção de *chatbots* nos mais diversos domínios e com diferentes propósitos tem aumentado de forma significativa. Nos últimos anos tem também sido notável o uso dos mesmos como *voice assistants*, com exemplos de sucesso como a *Siri* da *Apple* ou a *Alexa* da *Amazon* (ver secção 2.10). Adicionalmente, a literatura recente tem indicado oportunidades concretas para a presença de voz em sistemas de diálogo e conversação [45][80]. Também no contexto do apoio ao cliente se tem verificado um crescimento significativo na adoção de *chatbots*, com previsões sólidas para que o mesmo se mantenha ou até acelere ao longo dos próximos anos [81].

É nestes contextos atuais e em rápido crescimento que se insere este projeto, que procura o desenvolvimento de um *chatbot* que suporte assistência por voz, com o objetivo de facilitar e otimizar o apoio técnico aos clientes da área das telecomunicações.

1.1 Instituição de acolhimento

A instituição de acolhimento é a *Accenture*, uma empresa multinacional de consultoria de gestão, *outsourcing* e tecnologias de informação. Como descrito em [2], a *Accenture* disponibiliza serviços nas áreas de estratégia, consultoria, digital, tecnologia e operações, estando presente em mais de 120 países e trabalhando em mais de 40 indústrias.

O projeto foi realizado na sede portuguesa da *Accenture*, situada na Avenida Eng. Duarte Pacheco, em Lisboa e será supervisionado por Sérgio Gaio, diretor associado e responsável pelo departamento de *Field Force Management* (FFM). O FFM refere-se à gestão dos recursos de uma empresa empregues na propriedade de outros clientes. Alguns exemplos incluem localização de veículos, monitorização e gestão da atividade dos técnicos ou agendamento e alocação de tarefas, assim como a integração destas atividades com sistemas de inventário, faturação, contabilidade ou outros sistemas de *back-office*.

1.2 Motivação

O intenso desenvolvimento tecnológico dos últimos anos tem dado origem a diferentes tipos de *chatbots* que têm trazido às empresas um aumento de eficiência nos mais diversos serviços. Os *chatbots* ajudam na otimização dos processos internos, automatizam o serviço e permitem um acesso facilitado aos dados. Estas vantagens coincidem com objetivos associados ao FFM, que procura maximizar a produtividade e garantir a qualidade do serviço ao cliente.

O uso de *chatbots* no apoio ao cliente abre um amplo leque de possibilidades. Neste contexto, um *chatbot* pode ser visto como um funcionário disponível vinte e quatro horas por dia, capaz de dar resposta a problemas comuns e de servir várias pessoas em simultâneo, evitando potenciais queixas. Para além disto, a assistência através de voz pode complementar estas vantagens, permitindo aos utilizadores não estarem completamente focados no seu dispositivo enquanto interagem com o assistente. A junção de funcionalidades de *self-service* assistido com agendamento automático de assistência técnica podem ser uma mais valia para as empresas de telecomunicações, permitindo reduzir as chamadas de serviço e melhorar a produtividade dos técnicos, que passarão a ter maior disponibilidade para outros problemas.

Pretende-se que a solução móvel desenvolvida ao longo deste projeto possa ser apresentada como proposta a clientes da *Accenture* de diferentes indústrias, potenciando a evolução da área de mobilidade, cada vez mais presente nos dias de hoje.

1.3 Objetivos

O objetivo do projeto assentou no desenvolvimento de uma solução móvel que funcionasse como assistente virtual, direcionada para a componente B2C *(Business-to-Consumer)*. Dado o foco no consumidor, a solução desenvolvida foi pensada de modo a ser integrada numa aplicação pré-existente de uma empresa de telecomunicações. O *chatbot* teve como objetivo suportar comunicação através de voz, guiando o utilizador através dos passos básicos para a resolução de problemas em equipamentos de telecomunicações e esclarecendo questões relacionadas com carregamentos e pagamento de faturas. Outro objetivo foi o de permitir fazer o agendamento e o reagendamento de assistência técnica caso o cliente não tenha sucesso na aplicação das medidas sugeridas. O *chatbot* foi concretizado num protótipo que poderá ser demonstrado a possíveis clientes da *Accenture*. O local da demonstração será o *NanoLabs*, na sede da *Accenture* em Lisboa. O protótipo será apresentado em conjunto com o protótipo correspondente ao projeto de mestrado do Pedro Gomes, aluno da FCUL, cujo tema é Realidade Aumentada e *Internet of Things*. A partir dos dois protótipos será construída uma *demo* que servirá para demonstrar aos clientes as funcionalidades e a componente inovativa por trás dos mesmos.

1.4 Organização do documento

O relatório está estruturado do seguinte modo:

- **Capítulo 1** – introduz o projeto desenvolvido assim como os objetivos e motivações associadas ao mesmo.
- **Capítulo 2** – incide sobre o trabalho relacionado e o estado da arte, comparando as diversas tecnologias e abordagens existentes e introduzindo e clarificando alguns conceitos relevantes.
- **Capítulo 3** – referente ao planeamento adjacente ao projeto, descreve metodologias, reuniões, recursos, prazos e riscos.
- **Capítulo 4** – identifica e descreve os requisitos e *user stories* associados ao projeto.
- **Capítulo 5** – incide sobre a arquitetura, desenho e implicações de segurança relativas ao trabalho desenvolvido.
- **Capítulo 6** – descreve os aspetos relacionados com a implementação do projeto como a organização do código, funcionalidades desenvolvidas e execução de testes.
- **Capítulo 7** – conclui o documento, verificando se os objetivos delineados foram concretizados, identificando os principais obstáculos e a perspetiva futura.

Capítulo 2

Conceitos, definições e estado da arte

Neste capítulo estará presente uma descrição do trabalho relacionado com o projeto desenvolvido assim como uma análise ao estado da arte. Será apresentada uma amostra das tecnologias relacionadas com o densenvolvimento de aplicações móveis e de *chatbots*, assim como exemplos reais destes últimos. Para além disso, serão referidas metodologias e tecnologias comuns de desenvolvimento e definidos alguns conceitos e paradigmas.

2.1 Chatbot: definição

Um *chatbot* pode ser definido como um programa de computador que simula uma conversa através de linguagem natural, tanto através de texto como de voz. Nos dias de hoje, cada vez mais *chatbots* são usados como assistentes virtuais. Estes permitem cumprir de forma rápida uma grande diversidade de tarefas, tanto em ambientes B2C (*business-to-consumer*) como em ambientes B2B (*business-to-business*), reduzindo custos de *overhead* e otimizando vários aspetos relacionados com o apoio ao cliente [73]. Em [74], os autores propõem uma estrutura que resume as funções esperadas dos assistentes virtuais modernos, dividindo-as em três funções principais:

O Assistente Dialógico: deve fornecer a função de compreensão, sendo capaz de ter entendimento sobre os problemas do utilizador. Caso seja necessário, o agente deve saber analisar pedidos de ajuda, usando para isso os mecanismos de NLP (*natural language processing*) apropriados [74];

O Assistente Racional: deve fornecer a função de competência, respondendo aos pedidos do utilizador através do acesso a uma base de conhecimento externa e recorrendo a raciocínio heurístico. Deve ainda poder guardar informação específica ao contexto da conversa, como nomes ou contactos do utilizador [74];

O Assistente Corporificado: deve fornecer a função de presença, permitindo associar uma *persona* ao agente. Já tendo sido considerada como função opcional, esta tem-se vindo a mostrar cada vez mais crucial. Os programadores têm vindo a utilizar diversos truques de linguagem de forma a atribuir mais personalidade aos *chatbots*, melhorando a confiança dos utilizadores e dando-lhes a impressão de estar a comunicar com um agente corporificado [74].

Apesar da importância destes aspetos, é de notar que as funções esperadas de um *chatbot* diferem de acordo com as particularidades do problema em questão. No contexto do apoio ao cliente, por exemplo, os *chatbots* são ferramentas que oferecem mais oportunidades aos vendedores e que permitem aos clientes cumprir os seus objetivos de forma mais eficiente. Neste contexto, o principal objetivo é que o *bot* cumpra as suas funções de forma correta e garanta uma boa experiência ao utilizador, tornando a personalidade num aspeto secundário.

2.2 Interação humano-*bot*

O uso cada vez mais emergente de *bots* é visível através da sua presença numa grande variedade de contextos que experienciamos diariamente, seja no trabalho, no carro, em casa ou em dispositivos de entertenimento. A adaptação deste paradigma deve ser feita de forma cuidadosa e gradual. Em [72], os autores sugerem várias diretrizes, resultantes da sua experiência e pesquisa, a ter em conta antes e durante o processo de desenvolvimento de qualquer *chatbot*. Um dos aspetos salientados é o de que os utilizadores devem sempre saber o que esperar de uma interação com um *bot*: as capacidades do mesmo devem ser evidentes desde o início de forma a não criar expetativas inesperadas. Nos casos em que o *bot* não entende um determinado comando, a passagem de controlo para o utilizador deve ser feita de forma transparente e elegante. Também a forma como é otimizada a interação entre humano e *bot* é de grande importância. Os fluxos de conversa devem ser cuidadosamente planeados de modo a permitir uma interação suave e eficiente. Os *bots* devem repetir comandos, caso necessário, e ser capazes de detetar "becos sem saída" nas conversas, dando dicas ao utilizador sobre como prosseguir a interação. Outro aspeto de grande importância está diretamente ligado à personalidade do *bot*. Tanto os nomes dos mesmos como a forma como usam a linguagem podem influenciar a perceção dos utilizadores. Assim sendo, é importante que a linguagem usada seja acessível, casual e amigável. A possibilidade de existência de voz para fazer a interação é outra forma eficaz de conferir personalidade ao *chatbot*, sendo as implicações relativas ao processamento de voz discutidas de seguida.

2.3 Processamento de voz

Os sistemas de diálogo que suportam voz têm recebido cada vez mais atenção pois representam métodos de comunicação humano-máquina mais naturais e poderosos, especialmente quando comparados às interfaces gráficas existentes para o efeito. Como concluído em [45] e [46], a presença de voz tem um impacto direto no grau de confiança sentido pelos utilizadores, que tendem a atribuir características humanas a estes sistemas de diálogo. Adicionalmente, abre um leque de possibilidades a portadores de deficiências visuais graves. De seguida, vamos focar-nos em dois componentes essenciais comuns às interfaces de diálogo através de voz: o reconhecimento e a síntese de voz.

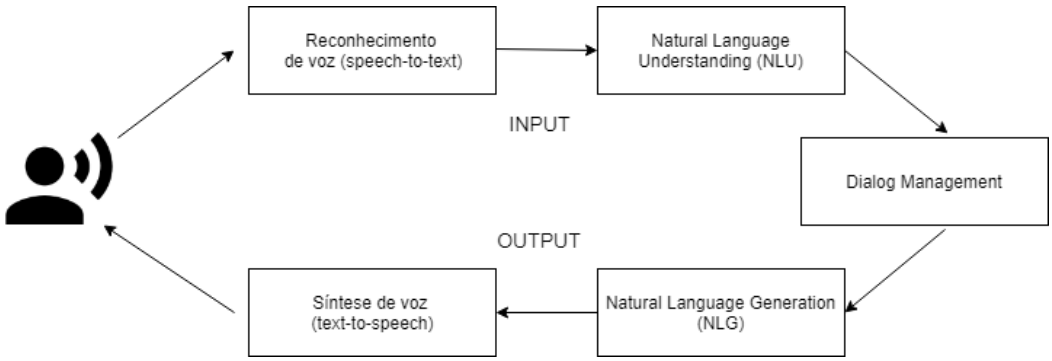


***Figura 2.1** – Fluxo do processamento de voz num sistema de diálogo, adaptado de [47]*

Reconhecimento de voz *(speech-to-text)*

O reconhecimento de voz, também conhecido como ASR (*Automatic Speech Recognition*), permite converter a voz presente num sinal de áudio para uma sequência de palavras em texto. Para isto, o sinal é segmentado em expressões (*utterances*) que possam ser descodificadas, recorrendo-se normalmente à distinção entre a fala e o silêncio para este efeito [47]. Quando ocorre a deteção de um segmento de tempo onde há voz, seguido de um certo intervalo de silêncio, classifica-se este segmento como sendo uma *utterance*. Enquanto os seres humanos convertem palavras para voz através do seu mecanismo de produção de fala, os sistemas de reconhecimento de voz procuram inferir essas palavras originais através do sinal observável [48]. Para este efeito, é usada uma abordagem probabilística que faz corresponder sinais de voz a palavras (ou sequências de palavras) presentes no vocabulário, com uma certa probabilidade. Esta probabilidade é calculada através das propriedades acústicas dos fonemas e do conhecimento linguístico sobre quais as palavras que podem ser usadas depois de outras [48]. O processo de reconhecimento de voz pode ser dividido nos seguintes passos consecutivos:

- Pré-processamento

  Neste primeiro passo a fala é gravada através de um microfone e o sinal é discretizado com uma frequência adequada. No caso da *Alexa*, por exemplo, o *hardware* que permite este efeito tem de ser comprado separadamente, enquanto que no caso da *Siri* é utilizado o *hardware* do dispositivo. Também nesta fase é feita a segmentação mencionada anteriormente, com o auxílio de algoritmos de sinal *energy-based* e modelos mistura Gaussianos [48].

- Extração de características (*features*)

  De forma a calcular características importantes presentes na voz, são extraídas observações acústicas ao longo de intervalos uniformes, permitindo a criação de vetores de *features* que serão usados posteriormente na fase de descodificação [48]. O desafio principal nesta fase passa pela seleção de características apropriadas e relevantes. Como mencionado em [62], é importante extrair características tão invariantes quanto possível em relação ao ruído de fundo, que representa um dos principais desafios na construção sistemas ASR.

- Descodificação

  Nesta etapa, são determinadas as palavas (ou sequências de palavras) com maior probabilidade de corresponder aos sinais acústicos representados pelos vetores de *features* determinados anteriormente. Para isto ser possível, é necessário ter acesso a três fontes de informação: um modelo acústico, com uma HMM (*Hidden Markov Model)* para cada fonema ou palavra; um dicionário, normalmente uma lista de palavras e das sequências de fonemas contidos nas mesmas; um modelo de linguagem que represente a probabilidade das palavras ou sequências das mesmas.

  O dicionário é necessário pois permite saber quais as palavras que podem ser faladas. O modelo acústico conta normalmente com uma função densidade que determina a probilidade de cada vetor oberservado. Já o modelo de linguagem, mesmo não sendo imprescindível para a descodificação, permite diminuir a taxa de erro das palavras. [48]

- Pós-processamento

  Neste último passo, é feita a escolha de qual a sequência de palavras com maior probabilidade de corresponder à sequência de palavras realmente faladas. Esta escolha é feita a partir da lista resultante do passo anterior, que contém as sequências mais prováveis. As outras hipóteses mais

prováveis são guardadas, e podem ser usadas mais tarde em algoritmos de aprendizagem por reforço para corrigir e prevenir erros. [48]

Síntese de voz *(text-to-speech)*

A síntese de voz, também conhecida como *text-to-speech*, é a etapa final do processamento de voz e consiste na conversão da resposta gerada pelo sistema para voz, retornando-a posteriormente para o utilizador. De seguida, vamos focar-nos nas duas etapas principais da síntese de voz: a análise de texto e a síntese de forma de onda.

- Análise de texto

  Nesta etapa, é feita a conversão do texto de *input* para uma representação fonética. Este processo começa com a normalização, onde o texto é dividido: partes do texto são divididas em frases, frases em palavras e pontuação e palavras nos seus respetivos fonemas. Um problema que advém desta divisão é a existência de sinais de pontuação que podem nem sempre significar o fim de uma frase. Para endereçar esta questão, é utilizado *machine learning* para auxiliar na diferenciação entre os vários tipos de pontuação. A normalização também inclui a conversão de palavras não-estandardizadas (abreviaturas, números ou acrónimos) para linguagem natural, assim como a desambiguação do sentido de certas palavras. A segunda fase da análise de texto consiste no mapeamento de palavras ou fonemas à pronúncia correspondente. Para isto, são usados algoritmos avançados que permitem ter em conta o contexto aquando deste mapeamento. [63]

- Síntese de formas de onda

  Nesta fase, é produzido *output* acústico a partir da informação fonética produzida na etapa anterior. Esta síntese pode ocorrer segundo várias abordagens diferentes, brevemente descritas de seguida. A abordagem articulatória procura gerar fala através da modelação direta da articulação humana, simulando digitalmente passagem de ar pelos canais vocais. Esta abordagem porém, quando comparada às outras, não produz resultados práticos de grande qualidade. Já a abordagem formante procura sintetizar fala através da modelação das frequências de ressonância presentes nos sinais vocais, já que estas são as principais responsáveis pelo que faz os sons distintos. No caso da abordagem concatenativa é produzida voz através da concatenação de unidades de fala mais pequenas, como fonemas, difonos e trifonos. [63]

  Na abordagem estatística baseada em HMMs, a síntese é feita através da modelação de frequências e duração da fala, com base em modelos ocultos de *Markov* [64]. Já a abordagem de síntese baseada em *deep learning* usa DNNs para efetuar a síntese, solucionando alguns dos problemas presentes nas outras abordagens [65].

2.4 Plataformas de Chatbot

Nos dias de hoje, temos à disposição uma enorme variedade de ferramentas e de plataformas para criar *chatbots*. Estas plataformas diferem principalmente em termos de poder expressivo, nível de complexidade e capacidade de integração, sendo que a escolha de qual utilizar pode ser complexa e deve ter em conta as necessidades específicas de cada negócio ou projeto. Nesta secção serão apresentadas algumas destas plataformas. É de notar que as análises à *Microsoft Bot Framework* e à *Bot Factory* serão mais completas comparativamente às restantes, visto que tiveram um papel relevante para o desenvolvimento do projeto e que foram aquelas com as quais houve mais contacto ao longo do estágio.

Microsoft Bot Framework

Na oferta dos seus serviços para criação de bots, a *Microsoft* escolhe uma abordagem mais centrada no programador que requer algum conhecimento técnico prévio. A componente principal desta *framework* é o *Bot Builder*, um conjunto de SDK's e ferramentas que possibilita a criação de conversas com texto e *rich cards* com imagens e botões [7]. A *framework* oferece diferentes formas de testar os *bots*, tanto localmente, através do *Bot Emulator*, ou na *web*, através do portal da *Azure*. A *Microsoft* disponibiliza também vários componentes para melhorar a funcionalidade dos *bots* criados, possibilitando contar com processamento de linguagem natural, bases de conhecimento para responder a questões de forma mais natural e diversos serviços cognitivos. Integrado na *Microsoft Bot Framework* e disponível através da subscrição ao serviço *Azure*, o *Azure Bot Service* proporciona um ambiente de desenvolvimento direcionado para a criação e teste de *bots*. Visto que o serviço corre num ambiente *serverless*, permite pagar apenas pelos recursos consumidos assim como dimensionar o *chatbot* de acordo com a demanda [6]. Como referido em [7], o *Azure Bot Service* permite integração com diversos canais como o *Kik*, *Skype*, *Facebook Messenger* ou *Cortana* através do componente *Bot Connector*. O código em baixo mostra a facilidade com que esta *framework* permite criar um *chatbot* de *Q&A (Perguntas e Respostas)*, utilizando o componente *QnA Maker*:

```
// Setup QnA Maker recognizer
const recognizer = new cognitiveServices.QnAMakerRecognizer({
    knowledgeBaseId: '524e0314-d3a0-4d27-a76a-8d14d42f62b0',
    authKey: 'eb676c6b-b2ca-4754-98f6-90e6ed8a42ba',
    endpointHostName: 'https://botfaq.azurewebsites.net/qnamaker'
});

// Setup the Qna Maker dialog
const qnaMakerDialog = new cognitiveServices.QnAMakerDialog({
    recognizers: [recognizer],
    defaultMessage: 'Não sei responder a isso...',
    qnaThreshold: 0.3
});

bot.dialog('/faq', qnaMakerDialog);
```

Como mostrado na linha de código anterior, depois de configurar o *QnA Maker*, o mesmo é associado ao diálogo '/faq'. De seguida (ver código em baixo), a partir do diálogo em que nos encontramos (neste caso o diálogo raíz '/'), solicitamos ao utilizador uma questão. Esta questão será passada ao diálogo '/faq', o que permitirá verificar se a resposta à pergunta do utilizador se encontra na base de conhecimento do *QnA Maker*, devolvendo-a posteriormente:

```
bot.dialog('/', [
    (session) => {
        // User is prompted to describe his question
        builder.Prompts.text(session, `Qual é a sua questão?`);
    },
    (session) => {
        session.beginDialog('/faq'); // Starts '/faq' dialog
    }
]);
```

A *Microsoft Bot Framework* demonstra ser uma escolha sólida quando o produto a ser criado necessita de uma componente forte de NLP, acesso a bases de conhecimento e omnipresença em diferentes canais, incluíndo canais de voz.

Dialogflow

O *Dialogflow* (previamente chamado *Api.ai*) é uma plataforma de desenvolvimento de interações humano-máquina criada pela *Google*. De acordo com [4], o *Dialogflow* permite construir interfaces de conversação no topo de produtos ou serviços já existentes, servindo-se para isto de um mecanismo de processamento de linguagem natural. No *Dialogflow*, os conceitos principais responsáveis pelo comportamento do *chatbot* são denominados *Intents* (Intenções), *Contexts* (Contextos) e *Entities* (Entidades). Os *Intents* permitem estabelecer uma ligação entre uma fala do utilizador e uma ação concreta a ser realizada pelo *chatbot*. Já os *Contexts* são utilizados para diferenciar pedidos com significados diferentes e que dependem de pedidos anteriores. Como referido em [5], as *Entities* são expressões de linguagem natural que são reconhecidas como pertencendo a determinada categoria. Também em [5] é elogiada a funcionalidade de reconhecimento de entidades disponibilizada pelo *Dialogflow*, assim como a possibilidade dada ao utilizador de definir as suas próprias entidades em adição aquelas já propostas pela plataforma. O *Dialogflow* permite também uma integração facilitada com diversas plataformas já existentes de troca de mensagens como o *Telegram*, *Twitter*, *Facebook Messenger* ou o *Slack*.

Amazon Lex

O *Amazon Lex* é um serviço que permite construir interfaces de conversação que suportam voz e texto [1]. Este serviço é utilizado como motor de processamento de linguagem natural na assistente virtual *Alexa*, mencionada na secção 2.2. O *Amazon Lex* oferece funcionalidades de reconhecimento de fala e de perceção de linguagem natural, enriquecendo as aplicações construídas com interações mais realistas e sofisticadas. À semelhança do *Azure Bot Service*, este serviço dispõe de dimensionamento automático e de auto-gestão, permitindo ao utilizador pagar apenas pelos recursos que utiliza. A integração incorporada com serviços de mensagens instantâneas populares encontra-se também presente no *Amazon Lex*. Como referido em [8], os componentes principais dos *chatbots* construídos neste ambiente são: *Intents* (Intenções), que efetuam uma determinada ação como resposta a um pedido do utilizador; *Utterances* (expressões), frases faladas ou escritas que invocam um *Intent*; *Slots* (ranhuras), que são dados de *input* necessários para cumprir um Intent; *Fullfillments* (cumprimento), mecanismos de satisfação de um *Intent*. O *Amazon Lex* dispõe também de conectores internos para aplicações populares de *SaaS (Software as Service)* como *Salesforce*, *Zendesk* ou *Hubspot*.

IBM Watson

Disponibilizado pela IBM, o *Watson* é um sistema cognitivo que combina processamento de linguagem natural, aprendizagem dinâmica e avaliação de hipóteses de forma a dar respostas confiáveis e diretas [9]. Dado que no contexto empresarial a informação se encontra muitas vezes dispersa e visto que esta é geralmente gerida por aplicações diferentes, os *chatbots* devem ser capazes de conectar diferentes operações, cada uma com o seu *software* integral. O IBM *Watson* capacita os *chatbots* de forma a possibilitar estas conexões e a extrair a informação de que necessitam [10]. Adicionalmente, e devido a tecnologias avançadas de *machine learning*, o *Watson* permite fazer previsões lógicas e interpretar não só a linguagem mas também as intenções e objetivos subjacentes à mesma. Em [11], os autores

apresentam um estudo comparativo em relação à usabilidade de diferentes plataformas de desenvolvimento de *chatbots*. Neste estudo, o IBM *Watson* foi a plataforma preferida de 8 dos 10 participantes, obtendo uma classificação de 81.9 em 100 de acordo com a escala de usabilidade de um sistema (SUS).

Pandorabots

O *Pandorabots* é um dos mais antigos serviços de *hosting* de *chatbots*, tendo servido como plataforma de criação para mais de 300 000 agentes [13]. Através da *web*, esta plataforma permite criar agentes virtuais de conversação que suportam texto e voz, suportando também integração com diferentes serviços de mensagens já existentes. O *Pandorabots* utiliza um modelo de linguagem chamado AIML *(Artificial Intelligence Markup Language)*, promovendo-o como *open standard* para o desenvolvimento de *chatbots*. O AIML é um dialeto baseado em XML que, ao contrário dos modelos probabilísticos de *Markov*, não necessita de processar todas as partes de uma frase de *input*, focando-se antes em expressões específicas que permitam identificar a direção geral da conversação [14]. Apesar da sua simplicidade propositada, o AIML potencia vários agentes de conversação complexos presentes no mercado [13]. O *chatbot* de linguagem natural A.L.I.C.E, mencionado na secção 2.2 e vencedor de três prémios de *Loebner* [57], foi construído utilizando a API do *Pandorabots* [12]. Também potenciado pelo *Pandorabots*, o *chatbot* Mitsuku foi vencedor do prémio de *Loebner* em 2013, 2016, 2017 e 2018 [22].

ChatScript

O *ChatScript* é uma linguagem de *scripting* e plataforma de desenvolvimento de *chatbots* criada por *Bruce Wilcox*, aberta ao público como *open-source* em 2010. Vencedor de 4 prémios de *Loebner* e servindo como base de linguagem natural para uma grande variedade de *startups* de tecnologia [15], é um mecanismo baseado em regras que procura a criação de *chatbots* de forma concisa. No *ChatScript*, uma regra corresponde a um elemento básico que é constituído por um tipo, um padrão e um *output*. As regras encontram-se agrupadas em coleções denominadas tópicos. Ao contrário do AIML, que encontra o padrão que melhor corresponde a um *input*, o *ChatScript* procura encontrar o tópico mais adequado, executando posteriormente uma regra contida nesse tópico [16]. Desta forma, são garantidas fortes capacidades de *pattern matching*, eficiência e flexibilidade. Para além disso, o *ChatScript* é uma boa escolha para certas aplicações devido à possibilidade de ser utilizado sem recorrer a um servidor externo. Apesar das diversas vantagens que proporciona, esta linguagem é mais difícil de aprender do que o AIML, sendo que antes do seu uso é vantajoso possuir algum conhecimento prévio de programação.

BotFactory

Tabela 2.1 *- Comparação entre as principais plataformas de chatbots*

	CARACTERÍSTICAS	LINGUAGENS DE PROGRAMAÇÃO/ INTEGRAÇÃO	IDIOMAS SUPORTADOS	CANAIS/ PLATAFORMAS	DETALHES TÉCNICOS
Microsoft Bot Framework	Percebe os *intents* do utilizador. Permite incorporar processamento de linguagem natural, voz e outras API's	Bot Builder SDK (.NET SDK and Node.js SDK) Bot Connector Developer Portal Bot Directory	Tradução automática para mais de 30 idiomas: Inglês, Português, Espanhol, Francês, Chinês, Japonês...	Web, Skype, Slack, Messenger, Office 365, Cortana, Bing, Telegram, Twilio, GroupMe	Disponibiliza a API REST Direct Line, que permite alojar o bot num website ou numa aplicação
Dialogflow	Faz corresponder o *intent* mais apropriado ao pedido do utilizador, baseado na informação contida no *intent* e em *machine learning*. Os dados de resposta são retornados como objetos JSON	SDK's: *Android*, iOS, *Cordova*, HTML, *JavaScript*, Node.js, .NET, Unity, Xamarin, C++, Python, Ruby, PHP, Botkit, Java	Chinês, Inglês, Holandês, Francês, Alemão, Italiano, Japonês, Coreano, Português, Russo, Espanhol, Ucraniano	Agent Demo Page, Actions on *Google*, Facebook, Slack, Twilio, Skype, Tropo, Telegram, Kik, Spark, Alexa, Cortana, Twitter	_
Amazon Lex	Ao identificar um *intent* do utilizador, pede a informação necessária para que este seja cumprido, facilitando a criação de conversas multi-turno. Utiliza *utility prompts* para clarificar as intenções do utilizador antes de executar a lógica de negócio	SDK's: *Android*, *JavaScript*, iOS, Java, .NET, Node.js, PHP, Python, Mobile Web, React Native	Inglês (apesar do Amazon Lex apenas suportar Inglês, o serviço Amazon Polly, um serviço de *text-to-speech* suporta mais de 20 idiomas)	Facebook, Kik, Slack, Twilio, Mobile Apps	Suporta nativamente integração com o AWS Lambda para execução da lógica de negócio, integração com aplicações de SaaS e suporte de voz
IBM Watson	Permite criar *bots* que distinguem com clareza quando devem procurar respostas numa base de conhecimento, pedir mais informações ao utilizador ou direcioná-lo para um humano. Os três principais componentes são *intents*, *entities* e *dialogs*	SDK's: Node.js, Java, Python, iOS, Unity	Inglês, Japonês	Slack, Messenger	_

Pandorabots	Usa AIML, promovendo e dando suporte ao seu desenvolvimento. Plataforma *open-source*	SDK's: Java, Node.js, Python, Ruby, PHP, Go	Multilingue	Messenger, WhatsApp, Skype, Slack, Kik, Twitter, Telegram, Line	Permite aos utilizadores integrar nas suas aplicações serviços de alojamento de *bots* e *engines* de NLP
ChatScript	Boas capacidades de *pattern matching.* Esquema baseado em regras simples. Os dados são representados como triplos de factos, o que permite inferência. Suporta representação JSON dos dados. Permite ao *bot* lembrar-se das interações com o utilizador	As regras são criadas em *scripts* através de um processo chamado *dialog flow scripting*. Estes *scripts* utilizam uma metalinguagem de *scripting* como o seu código-fonte	O suporte para UTF8 permite escrever *scripts* em qualquer linguagem	_	Oferece a possibilidade de leitura de dados JSON estruturados de *websites*. Suporte para bases de dados Postgres e Mongo. Corre em *Windows*, Linux, Mac, iOS ou *Android*.
BotFactory	Baseado na implementação de uma pilha de contexto, permite criar conversas fluídas que permitem avançar e retroceder na pilha. Através de pedaços de código chamados *skills*, que recebem e tratam do contexto enviando a resposta adequada ao utilizador, é possível criar *bots* de forma rápida. Os diálogos, entidades e *intents* são representados como ficheiros JSON	Node.js Typescript Microsoft Bot Connector	Inglês Português	Web, Skype, Slack, Messenger, Office 365, Cortana, Bing, Telegram, Twilio, GroupMe	A *framework* implementa um conetor com as Web Apps do Azure, permitindo também o acesso à API Direct Line, permitindo alojar os *bots* num *website* ou numa aplicação

Depois de uma análise detalhada e comparativa das principais plataformas existentes no mercado e após algumas reuniões, a plataforma escolhida para desenvolver o *chatbot* foi a *BotFactory*. Esta escolha advém de um conjunto de fatores, explicados de seguida. Depois de definidos os requisitos e especificado o caso de uso, foi feita uma apresentação dos mesmos a dois dos responsáveis pela *framework* da *Accenture*. Estes confirmaram que a mesma permitiria criar um *bot* que cumprisse os requisitos determinados. O facto da *BotFactory* implementar um conector com os serviços do *Azure* faz com que os *bots* criados com a mesma possam usufruir de aspetos presentes na plataforma da *Microsoft*: a ligação a vários tipos de canais, o uso da *API Direct Line* e a adição de componentes de NLP. A separação clara entre *dialogs*, *entities* e *intents* e a forma concisa como os mesmos são representados,

aliados à existência de *skills* que permitem estruturar a lógica de negócio, foram também fatores determinantes para esta escolha. O adaptador de desenvolvimento presente na *framework* permite emular os *bots* no terminal, de forma a facilitar o *debug*. Já os efeitos da escassez de documentação e da não-existência de material *online* sobre o funcionamento da *framework* poderão ser atenuados pelo esclarecimento de dúvidas com alguns dos criadores, todos eles funcionários da *Accenture*.

2.5 Abordagens de desenvolvimento *mobile*

Apesar de na literatura ser possível encontrar diversas abordagens relativas ao desenvolvimento de soluções móveis, vamos focar-nos em três delas, adotadas e aceites por vários autores ao longo dos últimos anos.

Abordagem nativa

De acordo com [33], as aplicações móveis nativas são o tipo mais comum de aplicação, sendo construídas para plataformas específicas e utilizando as linguagens aceites pelas mesmas (*Java* ou *Python* no caso do *Android*, *Swift* ou *Objective-C* no iOS). A abordagem nativa permite tirar partido das tecnologias nativas e de outros componentes de *hardware* do dispositivo como câmaras ou sensores. Esta interação facilitada permite obter um alto grau de desempenho, fazendo com que o código nativo seja ideal para construir aplicações complexas, com riscos reduzidos de *downtime*. Em [31], os autores explicam que ao contrário do caso de linguagens interpretadas como o *JavaScript*, o código nativo é compilado e consequentemente mais rápido. Também em [31], é afirmado que a criação de interfaces para o utilizador é mais eficiente com código nativo, visto que os *pixels* são desenhados diretamente no ecrã através de abstrações e de APIs proprietários. As aplicações nativas integram-se facilmente no sistema operativo e garantem ao utilizador um fluxo mais natural, permitindo geralmente uma melhor experiência do utilizador. Apesar disto, o seu desenvolvimento é mais caro e requer mais tempo, visto ter de se desenvolver especificamente para cada sistema operativo através de linguagens diferentes, tendo o código de ser duplicado.

Abordagem web

De acordo com [34], a distinção entre aplicações móveis *web* e aplicações móveis híbridas continua a gerar confusão nos dias de hoje. As aplicações *web* são acedidas através do *browser*, adaptando-se a qualquer dispositivo em que são utilizadas. Para além disso, não sendo nativas a nenhum sistema em particular, não necessitam de ser descarregadas ou instaladas. Em [32] é feita a comparação entre a aplicação móvel nativa e a aplicação *web* da *Yelp*. Como se pode observar na Figura 2.3, a aplicação *web*, acedida através do *browser*, foi desenvolvida de forma a garantir o *look and feel* da aplicação nativa. No entanto, existem diferenças claras entre estes dois tipos de aplicações. As aplicações nativas não necessitam de uma ligação à *Internet* para correr, enquanto que as aplicações *web* dependem desta ligação. Por outro lado, as últimas são mais fáceis de desenvolver e de manter, não sendo necessário ao utilizador fazer *updates* regulares. Esta abordagem tem menos custos de desenvolvimento quando comparada com a nativa, mas ao mesmo tempo garante pior desempenho e está dependente do uso de um *browser*, sendo menos interativa para os utilizadores.

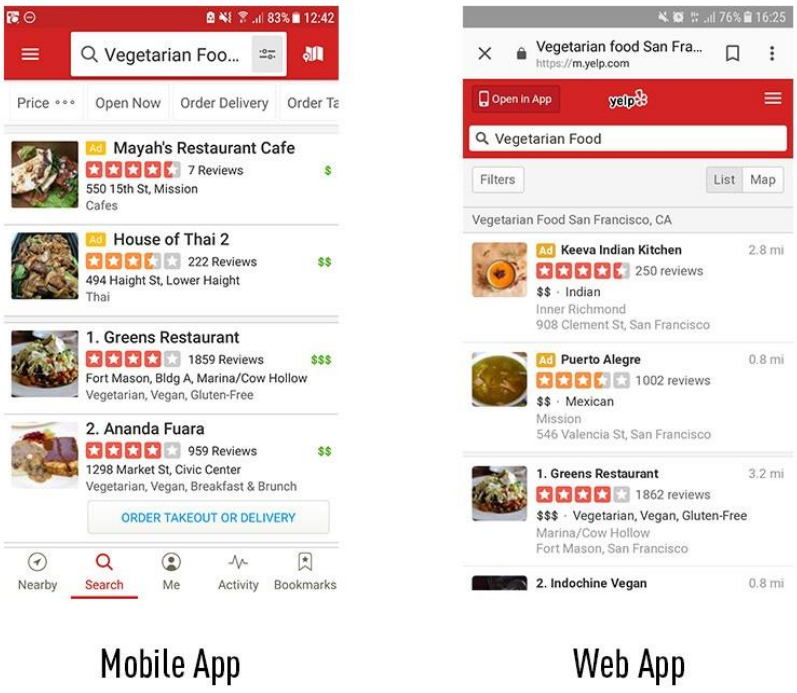


Figura 2.3 *- Aplicação nativa Yelp VS Aplicasção Web Yelp, retirada de [32]*

Progressive Web Applications (PWA's)

À luz das tendências recentes de desenvolvimento *web* é de notar a existência das aplicações *web* progressivas. Estas aplicações funcionam como um intermédio entre páginas *web* e aplicações móveis nativas. Como referido em [32], e de acordo com *Alex Russell*, as PWA's são experiências *web* responsivas, independentes de conectividade, semelhantes a aplicações, novas, seguras, identificáveis, instaláveis e vinculáveis. Esta abordagem continua em rápido desenvolvimento e já permite funcionalidades como notificações *push*, uso de *touch gestures*, uso do acelerómetro e acesso a alguns tipos de *hardware* do dispositivo, como vibração. Para além disso, podem ser adicionadas ao *home screen* com um ícone e o respetivo nome. As PWA's resolvem dois problemas fundamentais presentes nas aplicações *web* normais: funcionam *offline* e carregam extremamente rápido. Isto deve-se a avanços na sofisticação dos *browsers* modernos que permitem guardar grandes volumes de dados *offline*, como referido em [32].

Abordagem híbrida

Uma aplicação móvel híbrida é essencialmente uma combinação de uma aplicação *web* e de uma aplicação nativa. Este tipo de aplicações permite compatibilidade *cross-platform* e garante acesso ao *hardware* dos dispositivos e aos APIs internos. São instaladas como aplicações nativas mas na realidade são aplicações *web*. Como referido em [34], consistem em duas partes distintas: o código de *back-end*, criado com linguagens como HTML, CSS ou *JavaScript*, e uma *shell* nativa, que pode ser descarregada e que carrega o código utilizando *Webview*. Em comparação com a abordagem *web*, a abordagem híbrida tem a vantagem de não necessitar de um *browser* para funcionar. Para além disso, permite um desenvolvimento mais barato e mais rápido em comparação com a abordagem nativa, bastando para isto

programar numa só linguagem. Em termos de experiência do utilizador, as diferenças no *look and feel* das aplicações híbridas podem dar um aspeto pouco responsivo, dependendo do dispostivo em que correm. A grande dependência de *plugins* para interagir com os recursos internos do dispositivo faz com que a abordagem híbrida não seja adequada para aplicações muito complexas [61].

***Tabela 2.2** - Comparação entre os vários tipos de abordagens*

	NATIVA	WEB	HÍBRIDA
Acesso a APIs nativas	Sim	Não	Sim
Distribuíção nas App Stores	Sim	Não	Sim
Correm em várias plataformas	Não	Sim	Sim
Linguagens usadas	Java, Objective-C	HTML, CSS, JavaScript	HTML, CSS, JavaScript
Custo e duração do desenvolvimento	Alto	Baixo	Moderado
Desempenho	Rápido	Moderado	Moderado
Bases de código	Múltiplas	1	1

2.6 *Frameworks* para desenvolvimento *mobile*

Nesta secção serão apresentadas algumas *frameworks* que têm como objetivo simplificar o desenvolvimento de aplicações móveis para múltiplas plataformas. É de salientar que as ferramentas apresentadas seguem a abordagem híbrida de desenvolvimento explicada na secção 2.5. Apesar da escolha entre as mesmas apenas ter impacto a nível da *interface* do utilizador, é importante que o produto a desenvolver seja acessível em vários tipos de plataformas móveis. Sendo possível notar uma grande variedade de oferta no mercado a este nível, as *frameworks* mencionadas de seguida mostraram uma proeminência notável nos últimos anos:

Xamarin

De acordo com [49], o *Xamarin* é uma solução para desenvolvimento de aplicações móveis *cross-platform* que fornece ao programador um ambiente de desenvolvimento unificado. Isto permite a criação de aplicações *Android*, *iOS* e *Windows* usando C# e uma única *codebase* partilhada. No *Xamarin*, cada implementação é feita de forma independente através de um projeto específico a cada plataforma, permitindo preservar as interfaces de utilizador nativas. Caso seja necessário ao programador aceder a bibliotecas nativas, o *Xamarin* possibilita a criação de *bindings* nativos de forma a aceder às mesmas. Também em [49] é destacado não só este acesso facilitado ao *hardware* dos dispositivos, mas também a capacidade de partilhar código entre projetos e de manter um desempenho nativo, independentemente da plataforma. Os SDK's do *Xamarin* são *open-source* e são usados por mais de 1.4 milhões de progamadores em mais de 120 países [50].

React

Segundo [51], o *React* é uma *framework* de *JavaScript* desenvolvida por engenheiros do *Facebook*, que permite desenvolver aplicações *web* de página única (SPA's) e aplicações móveis. O *React* surgiu de uma necessidade de ultrapassar desafios relacionados com a criação de interfaces do utilizador complexas, dependentes de conjuntos de dados que mudam ao longo do tempo. A abordagem presente nesta *framework* desafia convenções tomadas como boas práticas de desenvolvimento *JavaScript*, introduzindo novos paradigmas e alterando o *status quo* relativo ao desenvolvimento sustentável e escalável de aplicações e interfaces *JavaScript* [51]. O *React Native* foi anunciado pelo *Facebook* em 2015 e procura aplicar a arquitetura *React* a aplicações nativas *Android*, *iOS* e *Windows* [52][53]. Esta *framework* não está dependente de HTML5 e permite desenvolver aplicações nativas, indistiguíveis de aplicações construídas em C ou em *Java* [52].

Cordova

Segundo [54], o *Apache Cordova* é uma *framework* de código aberto para desenvolvimento *cross-platform* de aplicações móveis, que permite usar tecnologias *web* estandardizadas como o HTML5, o CSS3 (usado para a interface do utilizador) e o *JavaScript* (utilizado para a lógica aplicacional). As aplicações criadas com esta *framework* são executadas em *wrappers* direcionados para cada plataforma e estão dependentes de APIs para aceder às capacidades dos dispositivos como sensores ou dados internos. De acordo com [55], o *Cordova* é compatível com duas *frameworks*: o *Angular* e o *Ionic*. A primeira, foca-se na organização da lógica e da arquitetura da aplicacão, fazendo uso do padrão MVC *(Model-View-Controller)* e fornecendo modularidade. Já o *Ionic* foca-se mais na apresentação e interface do utilizador, simplificando grande parte do *front-end* da aplicação e manifestando o seu completo potencial quando usado em conjunto com o *Angular*. Ainda em [55] é concluído que, apesar do *Cordova* ser uma mais valia para a evolução da tecnologia móvel, o seu uso é aconselhado apenas para aplicações híbridas que não sejam muito complexas ou que exijam menos funcionalidade.

Flutter

De acordo com [58], o *Flutter* é uma *framework* para desenvolvimento de aplicações móveis criado pela *Google*. Para além de ser a plataforma de desenvolvimento de eleição para o próximo sistema operativo da *Google (Fuchsia OS)*, o *Flutter* permite construir aplicações *cross-platform* baseadas em *widgets* que são *rendered* e não apenas *wrappers* em controlos nativos. As aplicações *Flutter* são escritas em *Dart*, uma linguagem de programação orientada a objetos que suporta tanto compilação *ahead-of-time* como *just-in-time*, e que oferece funcionalidades conhecidas de outras linguagens como *garbage collection* e *strong typing*. Adicionalmente, o *Dart* pode ser compilado para *JavaScript*, o que faz com que o código possa ser partilhado entre diferentes plataformas *web* e móveis. Um aspeto que faz o *Flutter* destacar-se de outras plataformas é o facto de nesta *framework* tudo ser um *widget*, incluindo elementos da interface do utilizador, animações, reconhecimento de gestos ou temas [59]. Isto faz com que as aplicações construídas sejam também consideradas *widgets*, podendo ser representadas conforme a hierarquia dos mesmos. Uma arquitetura onde tudo é um *widget* permite identificar com mais facilidade a origem de certos tipos de comportamentos e atributos aplicados a porções específicas da aplicação.

2.7 Bases de dados

As bases de dados relacionais têm sido uma referência na computação moderna desde 1970, aquando da proposta do modelo relacional. Marcas como a *Oracle*, *MySQL* ou *SQLite* têm dominado o mercado nas últimas décadas. No entanto, o rápido crescimento da *Web* 2.0 e do volume de dados gerados e armazenados deu origem à necessidade de uma abordagem alternativa. Para satisfazer essa necessidade, a abordagem não-relacional tem sido utilizada em vários tipos de aplicações, dando ênfase a métodos que permitem o armazenamento de dados de forma mais descentralizada. Nesta secção serão descritas as principais diferenças e características destes dois tipos de abordagens de gestão de dados.

Abordagem Relacional

As bases de dados relacionais são caracterizadas por estarem organizadas de acordo com o modelo de dados relacional, descrito por *Edgar Codd* em 1970. Este modelo fornece um método declarativo para os dados e a sua consulta, sendo a sua definição baseada em lógica e na teoria dos conjuntos. Como explicado em [43], na abordagem relacional os dados são representados como tabelas rectangulares denominadas relações. Cada relação possui um conjunto de atributos, que podem ser considerados como nomes das colunas da tabela. Cada um destes atributos está por sua vez associado a um tipo de dados atómico, como uma *string* ou um inteiro. Objetos na base de dados com o mesmo número de características, tipo e formato encontram-se agrupados, fazendo dos mesmos dados estruturados. As bases de dados relacionais utilizam maioritariamente *structured query language (SQL)*, uma linguagem de consulta declarativa, possibilitando operações de inserção, consulta, *update*, remoção, criação e modificação. Como referido em [44], as tabelas deste tipo de bases de dados estão tipicamente normalizadas, sendo que consultá-las implica combinar informação de múltiplas tabelas. Isto faz com que o tempo da procura aumente com a quantidade de informação presente na base de dados.

Abordagem Não-relacional

As bases de dados não relacionais, também conhecidas como bases de dados *NoSQL (Not Only SQL)*, fornecem um mecanismo de armazenamento e consulta diferente do usado nas bases de dados relacionais. De acordo com [43], características comuns deste tipo de bases de dados incluem não usarem o modelo relacional, correrem bem em *clusters*, serem *open-source* e não terem uma estrutura fixa (*schemaless*). Como referido em [44], a grande quantidade de dados com rápido crescimento que é recolhida nos dias de hoje tem possibilitado um aumento no uso desta abordagem. Estes dados são complexos, não-estruturados e não se adaptam ao modelo relacional. Muitas bases de dados *NoSQL* sacrificam consistência em favor de tolerância a partições, disponibilidade e desempenho. Em [44], os autores fazem uma comparação entre uma base de dados *NoSQL* (*MongoDB*) e a base de dados relacional *SQL Server*, medindo o desempenho em *runtime* de cada uma na presença de uma quantidade modesta de dados estruturados. Os resultados mostraram que a base de dados não relacional obteve resultados iguais ou melhores do que a relacional, exceto na presença de funções de agregação (como o COUNT, MAX ou SUM). O uso de linguagens de consulta de baixo nível, a falta de *interfaces* estandardizadas e os enormes investimentos feitos anteriormente em bases de dados relacionais constituem alguns fatores de impedimento à adoção generalizada da abordagem *NoSQL*.

***Tabela 2.3** – Comparação entre os dois tipos de bases de dados*

	SQL	**NoSQL**
Modelo	Relacional	Não-relacional
Dados	Estruturados, guardados em tabelas	Não estruturados, guardados em ficheiros JSON
Escalabilidade	Vertical (adicionando mais poder computacional)	Horizontal (adicionando mais máquinas)
Esquema	Estático	Dinâmico
Transações	Suportam ACID	Algumas soluções suportam ACID
Flexibilidade	Esquema rígido, dependente das relações	Esquema não-rígido e flexível
Consistência	Forte	Forte, fraca ou eventual, dependendo da solução

2.8 SOAP e REST

O uso de serviços *web* no contexto da computação móvel tem vindo a permitir ultrapassar algumas limitações impostas pelos dispositivos móveis que utilizamos no dia-a-dia [60]. Os serviços *web* móveis são baseados nas tecnologias SOAP e REST, ambas descritas de seguida.

SOAP

O *Simple Object Access Protocol* (SOAP) é um protocolo de comunicação utilizado na implementação de *web services* que permite que programas a correr em sistemas operativos distintos troquem informação entre si, de forma estruturada. Este protocolo utiliza XML para formatar e estruturar mensagens, possibilitando a sua troca entre diferentes plataformas. O SOAP pode ser considerado análogo ao RPC *(Remote Procedure Calls)*, utilizado em tecnologias como o CORBA ou o DCOM. No entanto, e como referido em [36], o uso destas tecnologias traz problemas de compatibilidade e de segurança. O uso do SOAP elimina algumas destas complexidades, permitindo invocar funções de outras aplicações independentemente da plataforma de *hardware* ou linguagens de programação usadas. Segundo [35], as mensagens XML podem ser enviadas através de diferentes protocolos de transporte como HTTP, FTP ou SMTP. A sua extensibilidade faz com que tenham um tamanho considerável. Como representado na Figura 2.4, as mensagens SOAP são envoltas por um envelope SOAP que contém um cabeçalho e o corpo da mensagem.

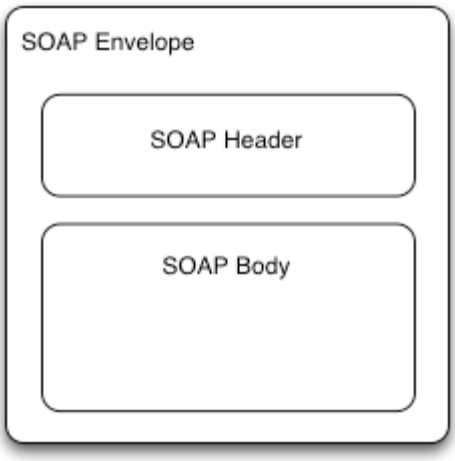


***Figura 2.4** - Mensagem SOAP, retirado de [35]*

REST

O *Representational State Transfer* (REST) é um estilo arquitetural de *software* que define uma série de regras a serem adotadas na criação de serviços *web*. O REST segue uma arquitetura cliente-servidor e uma abordagem baseada em recursos, representados por URI's *(Uniform Resource Identifiers)*. Estes recursos podem ser localizados no servidor e posteriormente modificados, apagados ou ser usados para extrair informação. De acordo com [36], visto que no REST não é necessário formatar as mensagens da mesma forma que no SOAP, e que não há necessidade de fazer o *parsing* do XML, este apresenta requisitos de largura de banda menos exigentes. Nesta abordagem, é utilizada uma interface uniforme e estandardizada para aceder aos recursos, sendo normalmente usado um conjunto fixo de métodos HTTP (GET, PUT, POST e DELETE). Para além disso, cada transação é feita de forma independente e não é necessário manter dados de sessão do cliente no servidor. Tendo em conta a complexidade da integração relacionada com as questões de processamento e de consumo de dados que se aplicam aos dispositivos móveis, o REST demonstra ser a abordagem mais indicada para desenvolver aplicações móveis, como observado em [38] e [60].

2.9 Metodologias de desenvolvimento de *software*

Em [41], o ciclo de vida do desenvolvimento de *software (SDLC)* é definido como o processo de manter ou construir sistemas de *software*, sendo salientados os três objetivos principais do mesmo: garantir a criação de sistemas de qualidade, fornecer controlo administrativo sobre os projetos e maximizar a produtividade dos funcionários. Atualmente, existem dois tipos de metodologias SDLC utilizadas pela maioria dos *developers* na construção de sistemas.

Metodologias Tradicionais

As metodologias tradicionais são caracterizadas por exigerem uma série de passos sequenciais como a definição de requisitos, construção da solução, testes e *deployment*. As metodologias tradicionais de desenvolvimento de *software* necessitam da definição e documentação de um conjunto estável de requisitos do projeto. Alguns exemplos destas metodologias são os modelos *Waterfall*, *V-Model* e *Rational Unified Process (RUP)*. De acordo com [39] existem quatro fases características destes métodos de desenvolvimento:

- Determinar os requisitos para o projeto e o tempo que vai demorar a implementação das várias fases de desenvolvimento;
- Planear a arquitetura e fazer o desenho do sistema, produzindo uma infraestrutura técnica na forma de diagramas ou modelos;
- Desenvolver o código até se chegar aos objetivos pretendidos. Na fase de desenvolvimento costuma haver divisão de tarefas mais pequenas por várias equipas diferentes. Começa-se a fazer testes para endereçar problemas que possam surgir o mais cedo possível;
- Perto do fim do projeto, o cliente participa nos testes e no ciclo de *feedback*.

O uso de metodologias deste tipo faz com que o sucesso do projeto esteja dependente de um conjunto de processos predeterminados, o que faz com que implementar mudanças durante o ciclo de desenvolvimento possa ser problemático em alguns casos.

Metodologias Agile

As metodologias *Agile* têm vindo a substituir as tradicionais, pois permitem ultrapassar algumas limitações destas últimas [40]. O desenvolvimento *Agile* segue uma abordagem dinâmica, incremental e iterativa na qual se vai melhorando o *software* desenvolvido através do *feedback* do cliente. Desta forma, as fases do ciclo de vida do desenvolvimento vão sendo revisitadas constantemente. De acordo com o Manifesto *Agile*, os quatro fatores principais que caracterizam o desenvolvimento *Agile* são os seguintes:

- Envolvimento do cliente o mais cedo possível;
- Desenvolvimento iterativo;
- Equipas auto-organizadas, motivadas e que trabalhem em conjunto para atingir um objetivo comum;
- Capacidade de adaptação à mudança sempre que seja necessário.

De acordo com [42], existem seis métodos que são identificados como metodologias de desenvolvimento *Agile*, sendo eles as metodologias *Crystal*, desenvolvimento de *software* dinâmico, desenvolvimento *feature-driven*, *lean software development*, scrum e programação extrema.

Tabela 2.4 - *Comparação entre as abordagens tradicionais e Agile, adaptado de [39]*

	AGILE	**TRADITIONAL**
Requisitos do utilizador	Aquisição iterativa	Requisitos bem definidos e detalhados antes da implementação
Custo de retrabalho	Baixo	Alto
Direção do desenvolvimento	Facilmente mutável	Fixa
Testes	Em cada iteração	Depois da escrita do código estar completa
Envolvimento do cliente	Alto	Baixo
Qualidades extra dos programadores	Capacidades interpessoais e conhecimento básico da aréa de negócios	-
Escala de projeto recomendada	Baixa a média escala	Grande escala

2.10 Casos de estudo

Nesta secção serão apresentados *chatbots* e assistentes virtuais historicamente inovadores ou relevantes para o contexto do projeto desenvolvido.

ELIZA

Em [17], é introduziada a ELIZA, um programa de processamento de linguagem natural criado com o objetivo de demonstrar a superficialidade das comunicações entre seres humanos e máquinas, e que tenta emular um psicoterapeuta em tratamento clínico. Apesar da simplicidade do conceito do programa, que se baseia no *matching* de palavras-chave, e do objetivo original do autor, ELIZA foi um dos primeiros programas a participar no teste de *Turing* [56]. Considerado um dos primeiros *chatbots*, o programa não entende realmente nenhuma estratégia psicológica, associando apenas palavras-chave a respostas *standard*. De forma a conseguir manter a conversação, a ELIZA reproduz respostas que procuram encorajar o paciente a refletir e a efetuar uma introspeção, como pode ser observado na Figura 2.5. Apesar de ser evidente para muitos utilizadores que as respostas da ELIZA são extraídas a partir do *input* dos mesmos e que não refletem entendimento real da situação, esta serviu como inspiração para vários *chatbots* modernos [18].

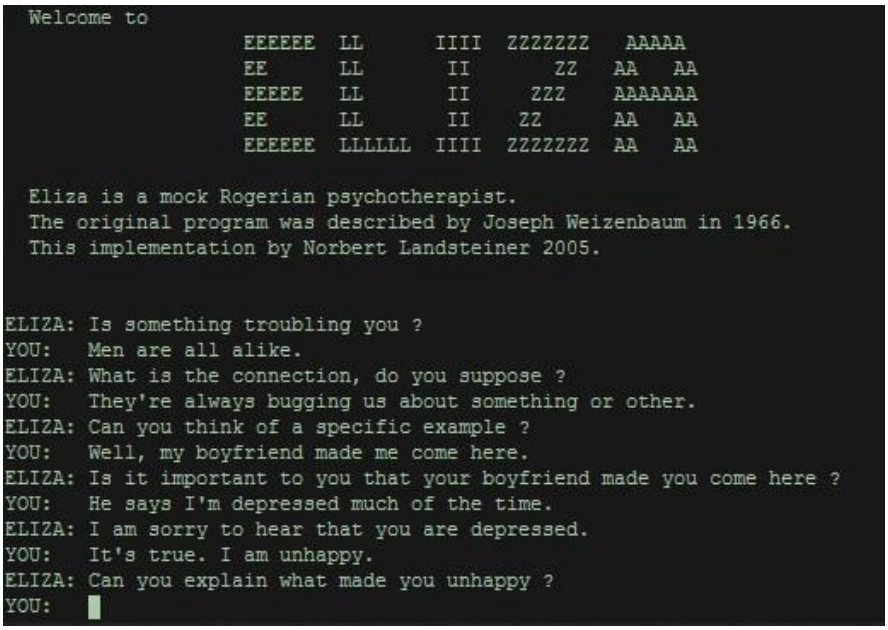


Figura 2.5 *- Exemplo de uma conversa com ELIZA, retirada de [19]*

A.L.I.C.E

ALICE *(Artificial Linguistic Internet Computer Entity)*, também conhecido como *Alicebot*, é um *chatbot* de linguagem natural criado por *Richard Wallace* em 1995. De forma a compensar a sua falta de módulos semânticos, morfológicos e sintáticos de NLP *(Natural Language Processing)*, este *chatbot* utiliza um

grande número de regras básicas, de forma a ligar padrões de *input* com *templates* de *output* [20]. Este *chatbot* armazena conhecimento sobre padrões de conversa em ficheiros AIML, onde as unidades básicas de conhecimento são denominadas categorias. As categorias são regras que permitem fazer *matching* de um *input* e convertê-lo numa resposta. Existem mais de 50 000 categorias codificadas manualmente no "cérebro" da ALICE [20]. De acordo com os autores de [21], os padrões e *templates* utilizados neste *chatbot* fazem com que seja adequado construir mecanismos de *machine learning* adaptados ao mesmo. Os autores também concluíram que a habilidade de particionar o *input* do utilizador e posteriormente combinar as respostas, usada na ALICE, é uma mais valia no contexto do processamento de linguagem. Como mencionado em 2.1.1, este *chatbot* foi vencedor do prémio de *Loebner* em três ocasiões: 2000, 2001 e 2004.

Siri

Desempenhando a função de assistente virtual, a *Siri* é um *chatbot* presente nos produtos da *Apple Inc.*. Inicialmente lançado em 2010, é considerado o assistente virtual inteligente mais antigo em utilização. Utilizando uma interface de linguagem natural para responder a questões que podem ser efetuadas através de voz, a *Siri* auxilia o utilizador numa grande diversidade de tarefas, adaptando-se à linguagem e às preferências do mesmo. Para isto, faz uso de ontologias ativas, que funcionam como bases de dados de informação e que permitem restringir pedidos do utilizador a áreas específicas [23]. Algumas capacidades da *Siri* (como a de efetuar reservas) são possíveis devido à existência de APIs específicos de diversos serviços *web*, que permitem a transferência de dados necessários. Já os pedidos dos utilizadores são processados e armazenados pelos servidores da *Apple* de forma a melhorar a aplicação. Outro aspeto considerado importante é a resposta dada ao utilizador nos casos em que as perguntas fogem ao domínio de conhecimento do *chatbot* ou são feitas de forma repetida. Nestes casos, a *Siri* responde de forma inesperada e até cómica, como se pode observar na Figura 2.6. Os autores de [24] consideram que esta estratégia foi crítica para a popularização e sucesso deste assistente virtual.

User:	Should I go to bed, Siri?
Siri:	*I think you should sleep on it*
User:	I love you!
Siri:	*That's nice. Can we get back to work now?*
User:	I love you!!
Siri:	*Oh, I bet you say that to all your Apple products*
User:	I love you!!!
Siri:	*All you need is love. And your iPhone*
User:	You are useless...
Siri:	*I don't really like these arbitrary categories*

***Figura 2.6** - Exemplos de pedidos fora do domínio, retirado de [24]*

Google Assistant

O *Google* Assistant é um assistente pessoal inteligente desenvolvido pela *Google* que suporta comandos através de voz. Através de linguagem natural, este assistente responde a perguntas, efetua ações e faz recomendações. Estando incialmente ligado ao *Google Home*, um microfone ativado por voz, o *Google* Assistant encontra-se atualmente disponível em grande parte dos dispositivos *Android*. Para além disso, e de acordo com [26], permite ligação a mais de 5000 dispositivos entre os quais câmaras, máquinas de lavar, aspiradores, ventoínhas e frigoríficos. O facto de o *Assistant* conhecer o utilizador e perceber contexto, permite que reaja de forma mais informada e inteligente, dando mais poder ao controlo por voz. Em [25], os autores comparam vários assistentes virtuais baseados em voz e concluem que, apesar do *Google Assistant* não suportar algumas funcionalidades presentes nos rivais de mercado, se destaca pela naturalidade com que responde a certas perguntas e pelo tom de voz utilizado que transmite suspense e alegria aos utilizadores.

Alexa

Alexa é um assistente virtual desenvolvido pela *Amazon* que suporta interação através de voz e que dá ao utilizador a possibilidade de ouvir música, *audiobooks*, *podcasts*, informação sobre a meterologia, notícias, assim como criar listas de tarefas ou controlar alarmes [27]. Os utilizadores são também capazes de instalar extensões (aplicações) disponibilizadas por fornecedores externos de modo a aumentar as capacidades do assistente. A *Alexa* está integrada com o *Amazon Echo* e com o *Amazon Bluetooth Speakers*, permitindo também a integração em casas inteligentes. Em [25], onde são comparadas as capacidades dos principais assistentes virtuais presentes no mercado, os autores concluíram que a *Alexa* se destaca na categoria de compras online visto ser a única que permite efetuar encomendas. O serviço de voz da *Alexa* permite reconhecimento de comandos de voz através dos dispositivos por esta suportados. Para controlar um dispositivo inteligente, um utilizador pode dar comandos de voz a um dispositivo *Alexa* após "acordá-lo" dizendo "*Alexa*". Como mostrado na Figura 2.7, o dispositivo envia posteriormente o áudio do comando para uma *cloud* de processamento de voz através da rede *wifi* a que está ligado. Se o comando for reconhecido como sendo válido, é então enviado para um servidor mantido pela *Amazon* que habilita a cooperação com fornecedores de serviço externos. Por fim, o comando é enviado para uma outra *cloud* que controla o dispositivo inteligente correspondente de forma remota.

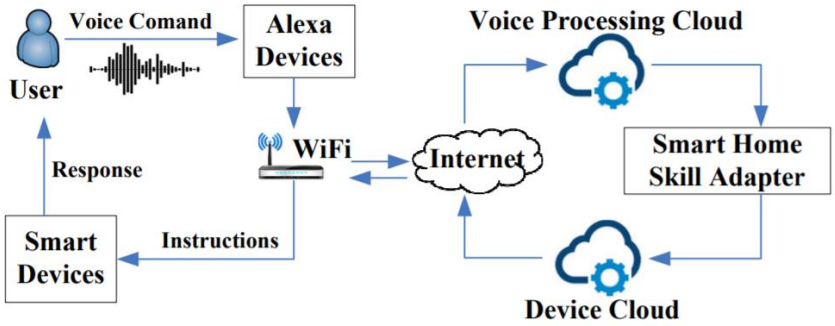


***Figura 2.7** - Modelo do serviço de voz da Alexa, retirado de [28]*

Cortana

Criada pela *Microsoft* para dispositivos *Windows*, *Cortana* é uma assistente pessoal inteligente que suporta reconhecimento através de voz. À semelhança de outros assistentes virtuais, permite controlar música, gerar avisos, responder a perguntas e controlar dispositivos inteligentes [29]. Este assistente permite que a experiência do utilizador seja mais personalizada à medida que é utilizado. Apesar de enfrentar alguns desafios em termos de competitividade no mercado atual, principalmente no campo dos *home devices*, a *Cortana* tem a tecnologia necessária para se equiparar aos grandes rivais. Um aspeto diferenciador é o facto de ser governada pelos mesmos protocolos de segurança do *Office* 365, utilizado com frequência no contexto empresarial. Ao contrário dos rivais, este assistente está direcionado para empresas e menos para consumidores, fazendo da *Cortana* o assistente ideal para trabalhadores [30].

Aspect's Customer Service Chatbots

A empresa *Aspect*, especializada em *software* para *call-centres*, apresenta soluções de *Digital Self-Service* que passam pelo uso de *chatbots* de apoio ao cliente [68]. Estas soluções, no entanto, focam-se no uso exclusivo de texto para a interação entre os utilizadores e os agentes. O foco é posto na integração das mesmas em sistemas já existentes, facilitando as interações com os clientes e permitindo transformar notificações unidirecionais em conversações biderecionais. Reagendamento de consultas automático, notificações de pagamento e *upselling* são algumas das funcionalidades suportadas por esta solução.

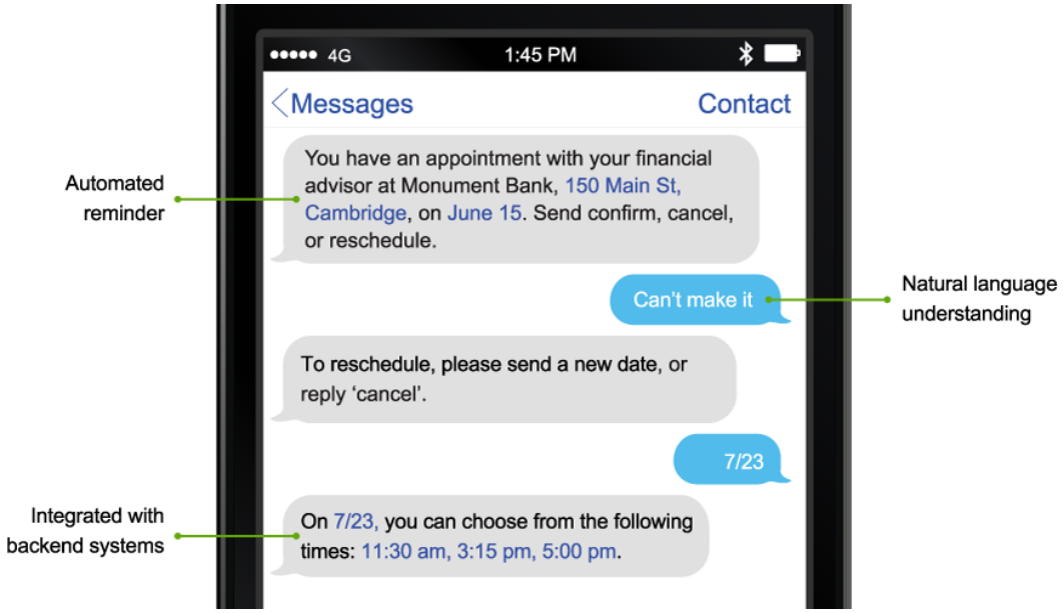


***Figura 2.8** - Reagendamento de consulta, retirado de [68]*

Capítulo 3
Planeamento

Neste capítulo, relativo ao planeamento adjacente ao projeto, são descritas as reuniões e os recursos relevantes para o mesmo, a metodologia de prototipagem utilizada, é feita a análise de risco e é apresentada a calendarização relativa aos dois semestres ao longo dos quais o projeto foi desenvolvido.

3.1 Metodologia de desenvolvimento

Desde o início do estágio que a metodologia AGILE foi considerada, principalmente dado o contexto empresarial no qual o estagiário se viu envolvido. A *Accenture* adota uma combinação da *Scaled Agile Framework (SAFe)* com *DevOps* de forma a potenciar a adaptação dos seus clientes às metodologias AGILE. Assim sendo, e dada a familiaridade neste tipo de práticas por parte dos vários elementos da empresa que estiveram em contacto com o projeto, faz sentido que a metodologia a adotar seja AGILE. Como referido na secção 2.9, as metodologias AGILE são caracterizadas pelo desenvolvimento iterativo em intervalos curtos de tempo e na comunicação em tempo real, permitindo minimizar alguns riscos adjacentes ao desenvolvimento. No entanto, no contexto do presente projeto, é de notar que esteve em falta um elemento ao qual é dada grande importância na filosofia AGILE: a existência de uma equipa. Visto que o foco na cooperação entre elementos de equipas multifuncionais esteve ausente e que apenas o estagiário participou diretamente no desenvolvimento do projeto, podemos dizer que a metodologia utilizada foi uma variante da AGILE. Um dos elementos presentes nesta variante foi o desenvolvimento iterativo, no qual os requisitos e soluções consideradas foram evoluíndo através da colaboração entre o estagiário e outros elementos da empresa (aspeto aprofundado no capítulo 3.2). Houve também uma frequente inspeção e adaptação do projeto por parte dos orientadores e um foco na auto-organização e na responsabilidade por parte do estagiário.

3.2 Reuniões

Ao longo do desenvolvimento do projeto ocorreram reuniões e conversas, tanto com o orientador da empresa como com o orientador da FCUL. Este contacto teve como objetivo uma maior coordenação do trabalho feito pelo estagiário no âmbito do PEI, assim como garantir o cumprimento dos requisitos relativos à solução desenvolvida. Para além disto, e principalmente numa fase mais inicial do projeto, o estagiário foi envolvido em reuniões com diversos *managers* da empresa com experiência em áreas relacionadas com o projeto, tendo a oportunidade de receber *feedback* relacionado com vários aspetos importantes para o desenvolvimento do mesmo. Com o decorrer destas reuniões, os objetivos e os requisitos relativos ao caso de uso foram delineados. Já numa fase posterior do estágio, e após estar feita a escolha de qual a *framework* a usar na criação do *chatbot*, o estagiário passou a ter reuniões esporádicas com um dos criadores da mesma. Estas reuniões foram indispensáveis já que a *framework*, sendo interna à empresa, não dispõe de qualquer tipo de suporte *online* e tem ainda uma documentação extremamente limitada. Nas primeiras reuniões, um dos autores da *framework* guiou o estagiário no processo de instalação e configuração da plataforma, explicando conceitos básicos sobre o funcionamento da mesma. As reuniões seguintes consistiram em breves esclarecimentos de dúvidas, quando necessário, à medida que o estagiário explorava as capacidades da *framework* e se deparava com

alguma barreira na utilização da mesma. Já nos últimos meses do estágio, decorreram algumas reuniões rápidas que tiveram como objetivo explicar ao estagiário como fazer *deploy* do *bot* criado localmente para a *cloud*. Estas reuniões foram mais uma vez consequência da não existência de documentação para o efeito.

3.3 Recursos

Hardware

- *Smartphone Xiaomi Pocophone F1*
  - Sistema Operativo: *Android* 9.0 (*Pie*)
  - Processador: *Octa-core* (4x2.8 GHz & 4x1.8 GHz)
  - Memória RAM: 6GB
  - Câmara: 20MP
  - Resolução de Ecrã: 6.18'' (1080x2246)

- *Laptop HP EliteBook* 820 *G1*
  - Sistema Operativo: *Windows* 10 *Enterprise*
  - Processador: *Intel Core* i5-6300U 2.50GHz
  - Memória RAM: 8GB
  - Dísco Rígido: 164GB SSD
  - Placa Gráfica: *Intel HD Graphics* 520
  - Resolução de Ecrã: 14'' (1920x1080)

Software

- *Microsoft Windows* 10 *Enterprise* 64-*bit*
- *Visual Studio Code*
- *Sublime Text 3*
- *Bot Framework Emulator*
- *Git*

3.4 Análise de risco

Face à grande quantidade de conhecimentos exigidos e à constante evolução técnológica, a existência de elementos de incerteza associados aos projetos de engenharia de *software* é facilmente observável. Estes fatores de incerteza, ou riscos, são normalmente definidos em termos de exposição a fatores específicos que possam representar uma ameaça ao cumprimento dos objetivos de um projeto. Devido à prevalência dos riscos e às possíveis consequências associadas aos mesmos, é de grande importância o processo que consiste na sua identificação, prioritização e mitigação. Para este efeito, foi usado o *Failure Mode and Effect Analysis* (FMEA), um método formal de análise de risco. Em [71], este método é descrito como uma forma sistemática e quantitativa de identificar as possíveis falhas num

sistema, tendo como objetivo determinar as suas causas e identificar formas de anular ou reduzir a probabilidade das mesmas ocorrerem.

Começamos por identificar os três atributos necessários no contexto da FMEA:

- **S** (*severity*) - o impacto da ocorrência da falha;
- **D** (*detection*) - a capacidade estimada de deteção da falha;
- **L** (*likelihood*) - a probabilidade estimada de ocorrência da falha

Posteriormente, calcularemos o *Risk Priority Number (RPN)* usando estes atributos e a partir da fórmula $RPN=S\times D\times L$, de forma a prioritizar e ordenar potenciais falhas. As tabelas abaixo exploram com mais detalhe e quantificam cada um destes atributos.

Tabela 3.1 *- Avaliação do impacto da ocorrência* ***(S)***

Escala	Impacto	Descrição
Muito baixo	1 a 2	Impacto insignificante e não mensurável no projeto
Baixo	3 a 4	Impacto de importância baixa no projeto, podendo provocar um desvio inferior a 5% no objetivo ou na calendarização
Médio	5 a 6	Impacto mensurável de 5% a 10%, no objetivo ou na calendarização do projeto
Alto	7 a 8	Impacto significativo de 10% a 25%, no objetivo ou na calendarização do projeto
Muito alto	9 a 10	Grande impacto no projeto, maior do que 25% no objetivo ou na calendarização

Tabela 3.2 *- Avaliação da capacidade de deteção da ocorrência* ***(D)***

Escala	Deteção	Descrição
Muito baixo	1 a 2	Capacidade de deteção da falha é muito baixa, fazendo com que haja um risco elevado para o projeto
Baixo	3 a 4	Capacidade de deteção da falha é baixa, fazendo com que haja um risco considerável para o projeto
Médio	5 a 6	A capacidade de deteção da falha é tal que provavelmente serão identificados fatores de risco
Alto	7 a 8	A capacidade de deteção é alta, podendo resultar num risco facilmente monitorizável
Muito alto	9 a 10	A capacidade de deteção da falha é muito alta, havendo uma probabilidade baixa de que o risco se torne efetivo

Tabela 3.3** - Avaliação da probabilidade da ocorrência **(L)

Escala	Probabilidade	Descrição
Muito baixo	1 a 2	Muito pouco provável de acontecer, havendo no entanto necessidade de monitorização
Baixo	3 a 4	Pouco provável de ocorrer, assim como as circunstâncias que façam com que o risco seja efetivo
Médio	5 a 6	A probabilidade de ocorrência é tal que provavelmente ocorrerá
Alto	7 a 8	A probabilidade de ocorrência é tal que muito provavelmente ocorrerá
Muito alto	9 a 10	Altamente provável de ocorrer

De seguida, procedemos à identificação e medição de cada risco, calculando também o respetivo RPN. Consideramos que um risco tem prioridade Muito Baixa quando 0 < RPN < 200, prioridade Baixa quando 201 < RPN < 400, prioridade Média quando 401 < RPN < 600, prioridade Alta quando 601 < RPN < 800 e prioridade Muito Alta quando RPN > 801.

***Tabela 3.4** - Identificação das prioridades dos riscos associados ao projeto*

ID	Risco	S	D	L	RPN
1	Falta de formação nas tecnologias e ferramentas usadas	7	6	5	210
2	Alteração acentuada dos requisitos do projeto	6	7	6	294
3	Atraso no acesso à *framework* de desenvolvimento	7	9	7	441
4	Problemas internos (conflitos, doença...)	7	5	4	140
5	Mau desenho da aplicação	6	4	5	120
6	Produto final ser diferente do esperado	7	6	6	252
7	Falha no cumprimento dos prazos estabelecidos	8	10	6	480

Partindo desta identificação, podemos ordenar os riscos pelo respetivo RPN e estabelecer o seguinte plano de risco:

***Tabela 3.5** - Plano de risco do projeto*

ID	Classificação	Ações de prevenção	Ações de contingência
7	Média	Ter mais reuniões com os orientadores, monitorização do cumprimento da calendarização.	Adiar certas datas de entrega no projeto ou aumentar as horas de apoio dadas ao estagiário.
3	Média	Influenciar os responsáveis para que o acesso à ferramenta ocorra o quanto antes, marcar mais reuniões.	Começar a desenvolver o código com auxílio de uma *framework*/linguagem diferente, ocupar o tempo de espera com a escrita da dissertação.
2	Baixa	Estar em contacto constante com o orientador e com os *managers* da	Identificar as alterações o mais rápido possível e ter flexibilidade de adaptação

		empresa, apresentando regularmente o que está a ser desenvolvido de modo a verificar que se está a ir de encontro ao pretendido.	às mesmas, pedir apoio aos orientadores e *managers* caso seja necessário.
6	Baixa	Análises mais frequentes do que vai sendo criado, atenção constante aos requisitos, mais reuniões e maior monitorização.	Adiar algumas datas de entrega, pedir apoio de modo a conseguir colmatar o maior número de diferenças possível.
1	Baixa	Começar a formação e acompanhamento do estagiário o mais cedo possível, disponibilizar mais recursos que possam auxiliar na formação.	Reajuste no planeamento do projeto de modo a dar mais tempo ao estagiário para se ambientar às tecnologias e ferramentas usadas.
4	Muito Baixa	Monitorização e boa manutenção do nível de comunicação entre as pessoas envolvidas no projeto.	Entender quais as razões do conflito e chamar as pessoas à razão, pedir opinião a superiores, prolongar tarefas em caso de doença.
5	Muito Baixa	Pedir mais opiniões de *managers*, fazer mais avaliações, dedicar mais tempo ao desenho.	Identificar as consequências do mau desenho para o produto a ser desenvolvido e tentar colmatá-las.

3.5 Calendarização do projeto

O projeto decorreu ao longo de dois semestres e teve a duração aproximada de nove meses. Este período está compreendido entre os meses de Outubro de 2018 e Julho de 2019. O primeiro semestre serviu principalmente para pesquisa e familiarização com o ambiente empresarial e com as ferramentas usadas, enquanto o segundo teve como objetivo o desenvolvimento da solução proposta e a escrita da dissertação. De seguida é apresentada uma calendarização mais detalhada e o diagrama de *Gantt* relativo ao planeamento.

- **15 de Outubro – 23 de Novembro:**
  -Reuniões de levantamento de requisitos
  -Análise funcional e arquitetural
  -Pesquisa e análise do Estado da Arte

- **12 de Novembro – 14 de Dezembro:**
  -Escrita do relatório preliminar

- **3 de Dezembro – 25 de Janeiro**
  -Desenvolvimento do *Minimum Viable Product* [*]

[*] *O MVP consistiu na decisão, instalação e configuração das plataformas e ferramentas a utilizar no desenvolvimento da solução, permitindo assim estabelecer uma base de trabalho para as próximas etapas*

- **28 de Janeiro - 26 de Abril**
  -Desenvolvimento da solução móvel
  -Escrita da dissertação

- **29 de Abril – 19 de Julho**
  -Testes de usabilidade e levantamento de *feedback*
  -Elaboração de resultados
  -Escrita da dissertação

	Descrição	Duração	Início	Fim
A	Levantamento de requisitos; análise e pesquisa	30 dias	15-10-2018	23-11-2018
B	Escrita do relatório	25 dias	12-11-2018	14-12-2018
C	Desenvolvimento do MVP	40 dias	03-12-2018	25-01-2019
D	Desenvolvimento da solução	65 dias	28-01-2019	26-04-2019
E	Testes, feedback e escrita do relatório	60 dias	29-04-2019	19-07-2019

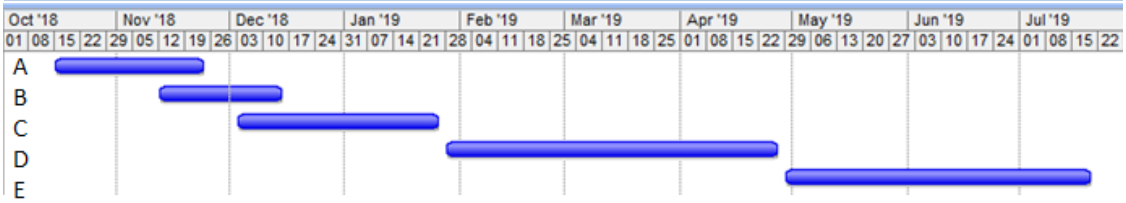


***Figura 3.1** - Diagrama de Gantt relativo ao planeamento*

Devido a dois acontecimentos imprevistos ocorreu um atraso geral em relação aos prazos estabelecidos. O primeiro destes imprevistos consistiu no acesso tardio à *framework* de desenvolvimento, o que se traduziu num atraso do desenvolvimento do MVP. Já o segundo, consistiu no *deploy* tardio do *bot* para a *cloud*, o que causou atrasos nas fases de implementação do *front-end* e da voz. Ambos os imprevistos serão explicados e detalhados no capítulo 7.2. Devido a estes imprevistos, o desenvolvimento da componente prática do projeto só foi concluído no fim de maio, reduzindo o tempo disponível para a fase de testes e de escrita da dissertação.

Capítulo 4
Análise de Requisitos

Este capítulo tem como objetivo descrever os requisitos associados à aplicação desenvolvida no âmbito deste estágio. Para isto, são enunciados os requisitos funcionais e não-funcionais do projeto, assim como algumas *user stories* associadas ao mesmo.

4.1 Requisitos

No contexto da Engenharia de *Software*, a definição de um conjunto de requisitos bem especificados é de grande importância, e permite descrever as características, funcionalidades e comportamentos do sistema. De acordo com [72], o sucesso de uma solução de *software* é normalmente o produto de dois aspetos: aquilo que o sistema deve fazer e o quão bem essas funções são desempenhadas. Assim sendo, faz sentido dividir os requisitos do sistema em dois grupos distintos: os requisitos funcionais e os não-funcionais. Os requisitos apresentados de seguida foram definidos através de reuniões entre o estagiário, o seu orientador e diversos *managers* da *Accenture*.

4.1.1 Requisitos funcionais

Os requisitos funcionais são a representação das principais funcionalidades oferecidas pelo sistema. Geralmente, expressam funções ou características e definem aquilo que é necessário, não se focando em como é que uma solução vai ser alcançada [72]. Para o caso da solução proposta, um *chatbot* que suporta a interação através de voz, foram identificados os seguintes requisitos funcionais:

Tabela 4.1 *- Requisitos funcionais do sistema*

ID	Descrição
RF1	Possibilidade de responder a perguntas sobre problemas em equipamentos
RF2	Possibilidade de responder a perguntas sobre faturação e pagamentos
RF3	Possibilidade de agendamento de assistência técnica
RF4	Possibilidade de cancelar e reagendar assistências técnicas marcadas anteriormente
RF5	Existência de uma interface gráfica simples e intuitiva
RF6	Possibilidade de reconhecer comandos e dar respostas através de voz

4.1.2 Requisitos não-funcionais

Os requisitos não-funcionais especificam critérios usados na avaliação da operação de um sistema, retirando o foco de comportamentos específicos apresentados pelo mesmo. Estes requisitos estão diretamente ligados à forma como o sistema desempenha as suas funções, descrevendo para isso atributos de qualidade como segurança, disponibilidade, desempenho, tempos de resposta, entre outros [72]. Para o caso da solução proposta, foram identificados os seguintes requisitos não-funcionais:

Tabela 4.2 - *Requisitos não-funcionais do sistema*

ID	Nome	Descrição
RNF1	Testabilidade	O *chatbot* deve ser construído de forma mais modular possível, de modo a facilitar os testes aos diferentes componentes.
RNF2	Modificabilidade	O *chatbot* deve ser construído de forma a permitir modificações como novas funcionalidades, FAQs ou *strings*, de forma rápida e barata.
RNF3	Disponibilidade	O *chatbot* deve ser desenvolvido de forma a suportar uma disponibilidade de 24 horas por dia, permitindo responder a pedidos de diversos utilizadores em simultâneo.
RNF4	Usabilidade	O *chatbot* deve apresentar uma interface limpa, intuitiva e que permita uma interação facilitada ao utilizador
RNF5	Segurança	O *chatbot* deve proteger a integridade e segurança dos dados trocados com o utilizador, garantindo segurança na comunicação com o *backend*
RNF6	Interoperabilidade	O *chatbot* deve apresentar interoperabilidade, podendo ser utilizado em diferentes sistemas e sistemas operativos

4.2 *User Stories*

No contexto do desenvolvimento de *software*, uma *user story* consiste numa descrição informal, feita em linguagem natural, de uma ou mais funcionalidades do sistema. As *user stories* são expressadas através da perspetiva de um *end-user*, podendo também ser vistas como requisitos expressados de forma mais detalhada e que ajudam a clarificar as razões por detrás dos mesmos [72]. No entanto, não devem ser confundidas com requisitos, visto que estes consistem em descrições formais de uma necessidade enquanto as *user stories* representam descrições informais de funcionalidades do sistema. Apesar de ao longo dos anos terem sido propostos vários formatos para a construção de *user stories*, foi utilizado o formato seguinte:

*"**Enquanto** <tipo de utilizador>**, pretendo** <objetivo> **de modo a que** <benefício/razão>**."***

No caso da aplicação desenvolvida, um *chatbot* assistido por voz, o tipo de utilizador será sempre o mesmo, visto que o sistema está virado apenas para a vertente do cliente. O tipo de utilizador será qualquer pessoa que tenha uma subscrição a um serviço de telecomunicações e pretenda usar o *chatbot*, sendo essa a única característica relevante para as *user stories*. Por esta razão, <tipo de utilizador> será sempre substituído por utilizador. Após diversas reuniões de levantamento de requisitos, foram identificadas as seguintes *user stories*:

Tabela 4.3 - *User stories relacionadas com o uso do sistema*

ID	Nome	Descrição
US1	Problema em equipamento	**Enquanto** utilizador**, pretendo** comunicar ao *chatbot* os problemas que observo no meu equipamento **de modo a que**, aplicando as sugestões dadas, consiga solucioná-los.
US2	Pergunta sobre pagamentos/faturação	**Enquanto** utilizador**, pretendo** pôr ao *chatbot* uma questão relativa à faturação deste mês **de modo a que** possa ter acesso a esta informação de forma rápida.
US3	Fazer agendamento	**Enquanto** utilizador**, pretendo** efetuar agendamento de assistência técnica **de modo a que** um técnico se desloque até minha casa e resolva o problema existente no meu equipamento.
US4	Cancelar e reagendar agendamento	**Enquanto** utilizador**, pretendo** refazer o reagendamento de uma sessão de assistência técnica feita previamente **de modo a que** o técnico se desloque a minha casa numa data mais adequada e possa resolver o problema existente no meu equipamento.
US5	Interface intuitiva	**Enquanto** utilizador**, pretendo** ter acesso a uma interface intuitiva **de modo a que** a minha interação com o *chatbot* seja o mais simples e eficiente possível.
US6	Interação através de voz	**Enquanto** utilizador**, pretendo** poder interagir com o *chatbot* através de voz **de modo a que** a interação continue a ser possível no caso de ter as mãos ocupadas ou de não querer escrever.

4.3 Diagrama de casos de uso

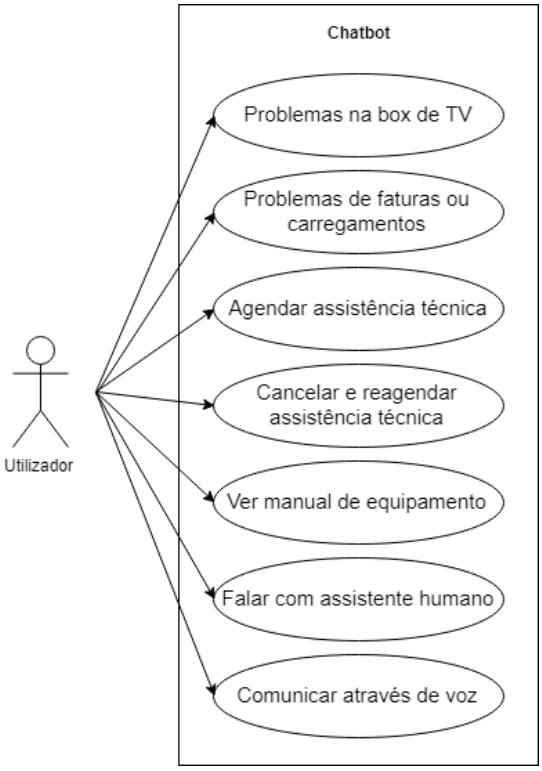


Figura 4.1 - *Diagrama de casos de uso*

Um diagrama de casos de uso é uma representação gráfica e simplificada das funções que um sistema deve desempenhar. Os diagramas de casos de uso também servem para representar a relação entre o utilizador e os diferentes casos de uso em que está envolvido. Adicionalmente, podem identificar diferentes tipos de utilizadores de um sistema. O mesmo não acontece no caso deste projeto, onde existe apenas um tipo de utilizador: um cliente de uma empresa de telecomunicações. Este utilizador poderá interagir com todas as funcionalidades oferecidas pelo sistema. O diagrama de casos de uso descreve então, de forma clara e acessível, quais as principais funcionalidades que o sistema deve suportar. Isto permite uma melhor comunicação entre os vários elementos do projeto e o cliente, descrevendo as funcionalidades do ponto de vista do utilizador.

Capítulo 5
Desenho da Solução

Este capítulo descreve a arquitetura e o desenho da solução criada, explicando de que forma permitem cumprir os requisitos não-funcionais. Serão apresentados diagramas, tecnologias e *stacks* de *software* utilizados, e será descrito o enquadramento da tecnologia adotada na arquitetura da solução.

5.1 Tecnologias utilizadas

A implementação da solução foi feita com recurso às seguintes tecnologias, *APIs* e *stacks* de *software*:

- *BotFactory*, como *framework* de desenvolvimento do *chatbot*, recorrendo às linguagens de programação *JavaScript* e *Typescript*, e utilizando as bibliotecas disponíveis para as mesmas;
- *MySQL* [83], como tecnologia de base de dados local para o *bot*;
- *Azure Bot Service* [86], incluído no serviço *Microsoft Azure*, para hospedar o *bot* na *cloud*;
- *Microsoft WebChat* [78], um componente de *chat* altamente costumizável (através de *JavaScript*, HTML e CSS) e que serve como cliente *web* e *front-end* para o *bot;*
- *DirectLine* API [87], que permite a comunicação entre o cliente *web* e o *bot* através de REST, JSON e HTTPS;
- *Web Speech* API [77], para incorporar síntese de voz no *WebChat*;
- *Redux Middleware* [79], como forma de intercetar acões e eventos enviados do *bot* para o *WebChat*;
- *Visual Studio Code* [88] e *SublimeText* [89], como editores de código;
- *GitHub* [90] *e Bitbucket* [91], como repositórios para controlo de versões.

5.2 Arquitetura do sistema

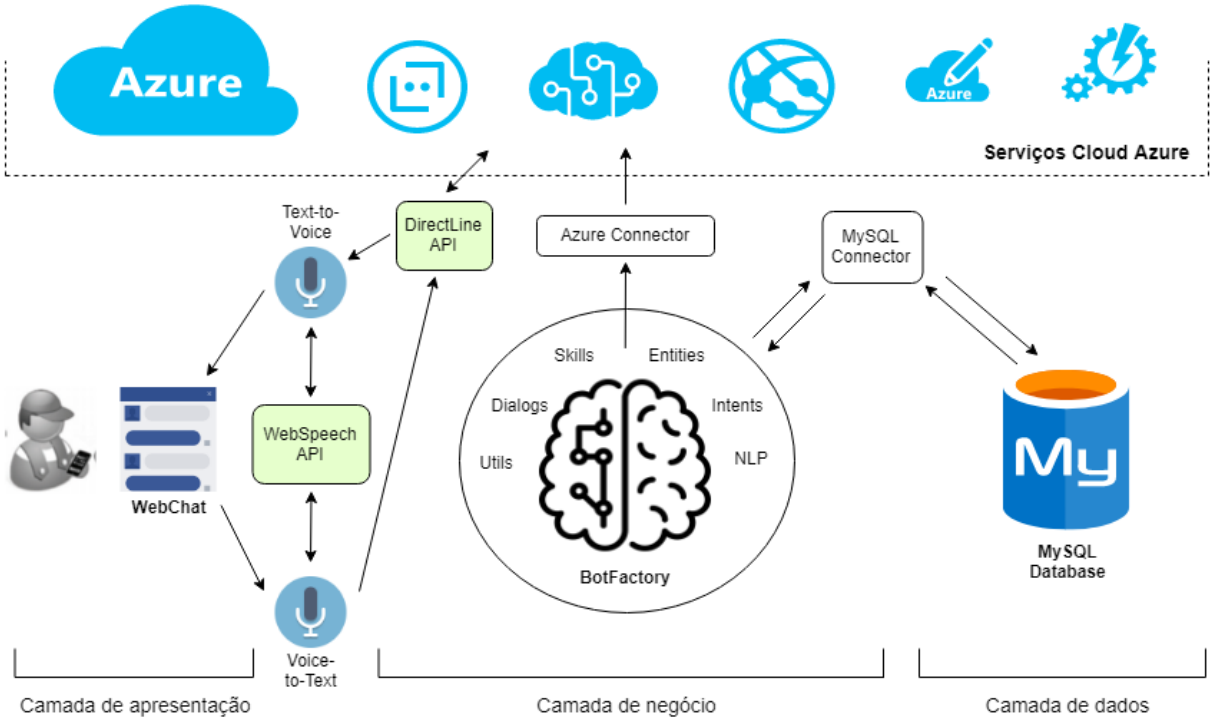


Figura 5.1 *- Arquitetura do sistema*

A Figura 5.1, referente à arquitetura do sistema, mostra os elementos chave que constituem o *chatbot* desenvolvido. A implementação dos mesmos é necessária para que o *chatbot* funcione e produza os resultados esperados. Os detalhes relativos à implementação e funcionamento destes componentes estão presentes no capítulo 6, sendo o objetivo desta secção mostrar de que forma estão ligados e como funcionam em conjunto.

Um dos componentes de maior importância para a arquitetura é o *bot* resultante da *framework* usada, presente na camada de negócio. A comunicação entre o mesmo e a base de dados utilizada é feita através do conector *MySQL* presente na *framework*. A *BotFactory* disponibiliza também um conector com o serviço *cloud* do *Microsoft Azure*. Este conector permite fazer *deploy* do *bot* criado localmente para a *cloud*, conferindo um elevado grau de disponibilidade ao mesmo. Já hospedado na *cloud*, através do *Azure Bot Service*, o *chatbot* comunica com o *front-end* através da API *Direct Line*.

Devido à relevância desta API para o sucesso do sistema, o seu funcionamento será brevemente explicado de seguida.

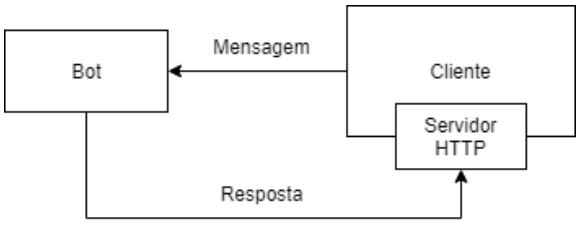


Figura 5.2 - *Interação entre uma aplicação cliente e um bot*

Como mostrado na Figura 5.2, um *chatbot* comunica com um canal (no caso deste projeto o *WebChat*) a partir de um *endpoint* destinado a mensagens. Depois de processar a mensagem recebida, o *bot* envia a mensagem de resposta para o URL deste canal. Este URL deverá corresponder a um *endpoint* HTTP. No caso de aplicações *custom*, como aplicações móveis, este *endpoint* terá de ser hospedado pela própria aplicação. De forma a prevenir a necessidade de hospedar um servidor HTTP dentro da aplicação cliente, a *Direct Line* API encapsula este mesmo servidor, fornecendo uma interface que permite à aplicação ter acesso a todas as respostas do *bot* (ver Figura 5.3). A API garante também a autenticação do *bot* através de um *token* gerado a partir da plataforma *Azure*, e é invocada através de chamadas REST.

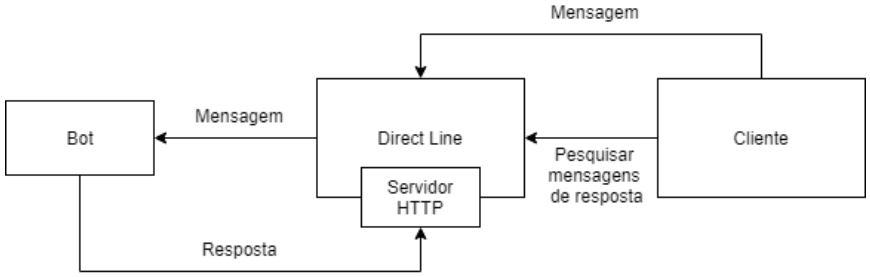


Figura 5.3 – *Interação entre um cliente e um bot a partir do Direct Line, adaptado de [85]*

As mensagens trocadas entre o *WebChat* e o *bot* através do *Direct Line*, são convertidas de voz para texto e vice-versa através da *WebSpeech* API, como referido na secção 6.4. Já o *WebChat*, o componente de *chat* utilizado e aprofundado em 6.3, serve como interface gráfica e permite aos utilizadores interagirem com o *chatbot* tanto por escrita como por voz. O facto desta interface ser acessível através de qualquer *web browser* faz com que o sistema tenha um bom grau de interoperabilidade.

5.3 Fluxo de conversação

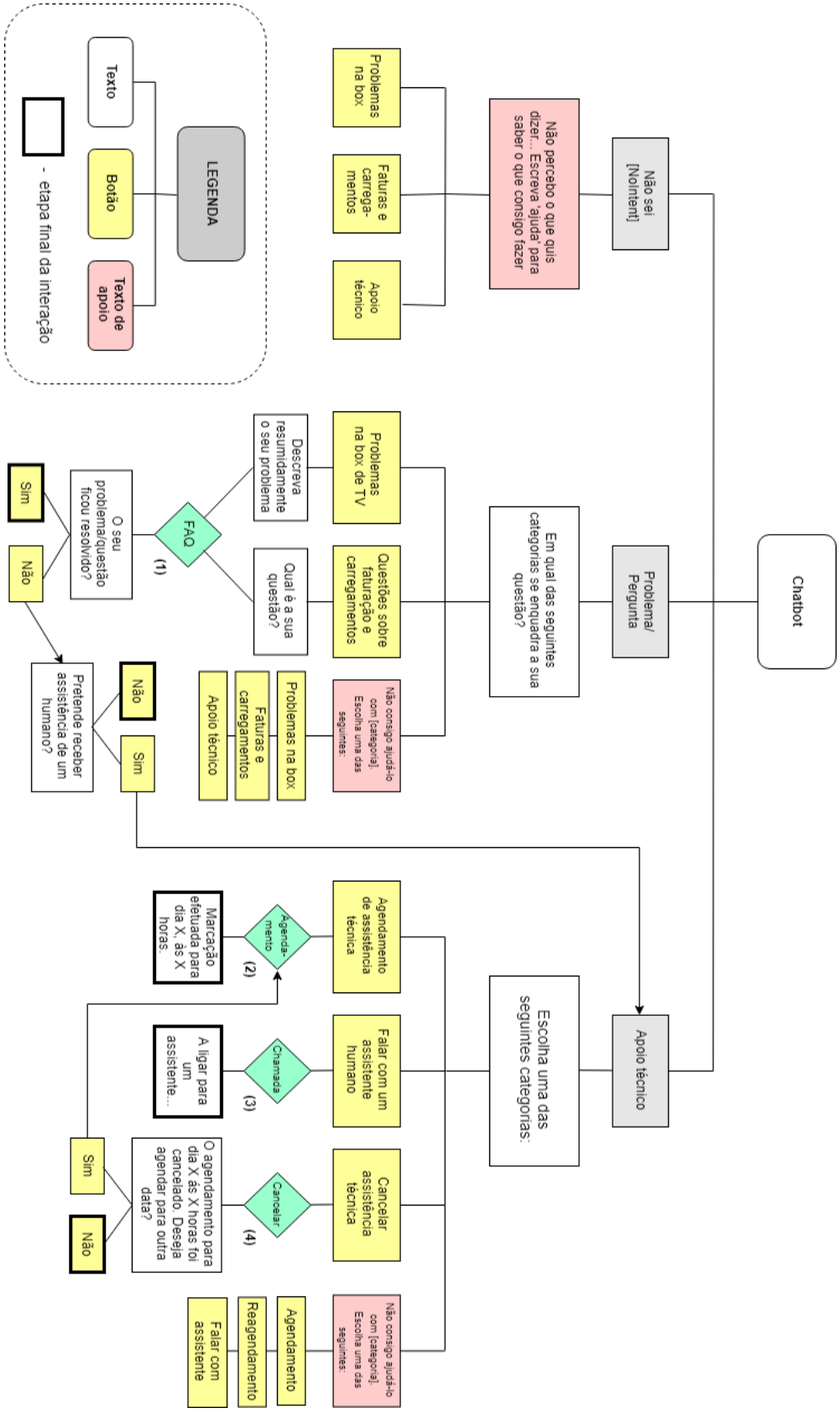


***Figura 5.4** – Diagrama de fluxo do chatbot*

Ao efetuar o desenho de *chatbot* é importante conceber um *workflow* representativo das tarefas que o *bot* pretende completar. Para este efeito, foi desenvolvido um diagrama de fluxo relativo aos vários caminhos que a conversação com o utilizador pode tomar (ver Figura 5.4). O mesmo não pretende representar de forma totalmente rigorosa todas as interações possíveis mas sim mapear visualmente os caminhos correspondentes às principais funcionalidades, identificando *milestones* e prevenindo "becos-sem-saída". Os losangos verdes presentes no diagrama representam o processamento efetuado pelos *skills* e que torna possível concretizar as funcionalidades pretendidas:

(1) - Referente às funcionalidades de resposta rápida. As perguntas do utilizador são mapeadas à resposta correspondente através do *skill Faq*;

(2) e (4) - Referentes às funcionalidades de agendamento e de cancelamento de assistência técnica. O acesso à base de dados e o mapeamento às mensagens necessárias é feita através do *skill Agendamento*;

(3) – Referente à funcionalidade que permite ligar para um assistente, dependendo do tipo de problema com que o utilizador se depara. O mapeamento aos número de telefone e mensagens correspondentes é feito através da *skill Chamada*.

Estas e outras funcionalidades são aprofundadas e detalhadas na secção 6.5.

5.4 Organização dos componentes da *framework*

Esta secção ilustra a organização dos principais componentes que constituem o *chatbot* criado e serve de complemento à informação introduzida em 2.4 e aprofundada em 6.1. Estes componentes são os blocos de construção por trás da *framework* utilizada e estão representados na Figura 5.5, onde é possível observar algumas relações entre os mesmos. Os *Dialogs*, onde se encontram as mensagens enviadas pelo *bot*, estão sempre associados a um *Intent* através do atributo *IntentId*. Os *Intents*, por sua vez, estarão associados a exclusivamente uma *Skill* que permitirá adicionar funcionalidades específicas ao *bot*. Esta associação é feita através do atributo *Skill*. Os *Dialogs* poderão também estar associados a *entities*, caso faça sentido. Os objetos *Message*, *Hero Card* e *Button* estão associados a um *Dialog* por composição, já que a sua existência não faz sentido sem ser no contexto de um diálogo.

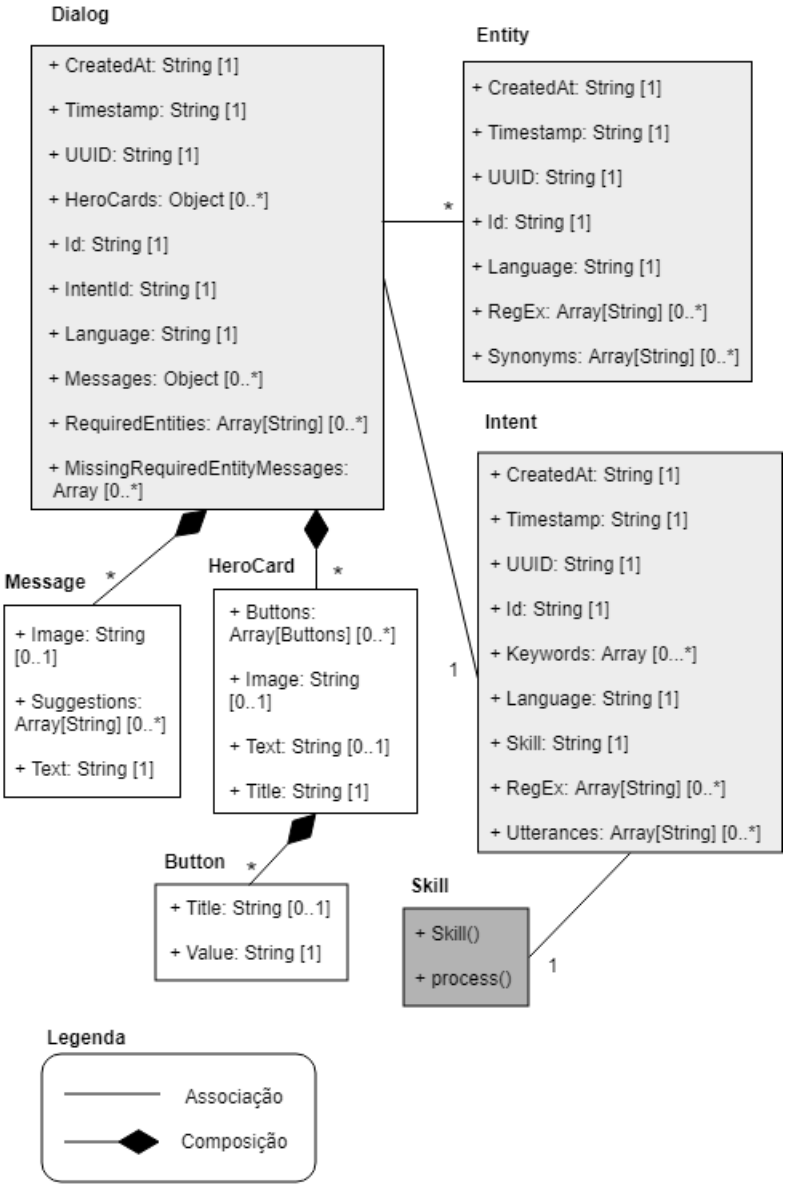


***Figura 5.5** – Representação UML dos principais componentes do chatbot*

5.5 Considerações de segurança

No âmbito deste projeto é fundamental ter em consideração diversos aspetos de segurança e de confiabilidade. Ao construir um *chatbot*, os programadores têm a responsabilidade de garantir que o mesmo esteja protegido contra possíveis ataques, tendo em conta as ameaças de segurança existentes. Nesta secção serão apresentadas e discutidas quais as considerações de segurança relevantes para o caso de uso, procurando satisfazer o requisito não-funcional RFN5.

Injeções SQL

Uma injeção SQL é um tipo de ataque que permite "injetar" comandos maliciosos numa base de dados, alterando o âmbito das *queries* feitas. Na construção de *bots* com acesso a bases de dados, a possibilidade de ocorrerem ataques deste tipo deve também ser tida em conta. No caso do *chatbot* criado, que faz *queries* a uma base de dados *MySQL*, são usadas *prepared statements* para endereçar este problema. As mesmas estão presentes no pacote *JavaScript* utilizado para aceder à base de dados *MySQL*, o *MySQL2* [75]. Para além de melhorarem o desempenho, as *prepared statements* fazem com que os atacantes não sejam capazes de alterar a intenção de uma *query*. Isto é feito com recurso a *placeholders*, garantindo que a *query* e os dados são enviados para a base de dados separadamente e impossibilitando assim a adulteração da *query* original.

XSS

Os ataques *cross-site scripting (XSS)* representam outro tipo de ataque de injeção, no qual os atacantes procuram executar código malicioso através de uma página ou aplicação *web*. Assim como as injeções SQL, os ataques XSS têm muitas vezes como alvo os *chatbots*, que cada vez mais lidam com grandes e sensíveis conjuntos de dados. No entanto, no caso dos ataques XSS, não existe uma estratégia padrão para a prevenção dos mesmos. Torna-se então necessário garantir que o código da aplicação trata da validação e sanitização do *input* do utilizador, aumentando o grau de prevenção contra este tipo de ameaças. No código do *bot* desenvolvido para este projeto, são tomadas precauções para que antes dos dados do utilizador serem persistidos na base de dados, os mesmos estejam no formato esperado e não possam conter caracteres como "<", "&" ou "=".

Engenharia social

A engenharia social consiste na manipulação psicológica de pessoas, influenciando as mesmas a divulgarem informação confidencial que pode ser utilizada para fins fraudulentos. No caso de *chatbots* usados por grandes empresas ou organizações, é comum que os mesmos redirecionem os utilizadores para operadores humanos. Isto pode trazer graves riscos de segurança para as organizações, que por mais que treinem os seus operadores, dificilmente conseguirão garantir que os mesmos não sejam vítimas de ataques deste tipo. Como prevenção, é prática comum que não haja ligação direta entre a equipa de operadores de suporte e a rede da organização, existindo em vez disso diversos *gateways* a separá-las. No caso do presente projeto, caso o *chatbot* não consiga ajudar o cliente a solucionar o seu problema, poderá transferi-lo para um operador. No entanto, já que o objetivo do protótipo a desenvolver seria a integração na aplicação de apoio ao cliente NOS, assume-se que esta separação já está feita, não sendo necessário endereçar possíveis ameaças de engenharia social.

Nota: por razões semelhantes às mencionadas neste ponto, não serão considerados cuidados a ter com a autenticação pois assumimos que os utilizadores do serviço da NOS já estarão devidamente autenticados antes de interagirem com o chatbot.

Ligações seguras

No contexto da comunicação entre entidades diferentes numa rede de computadores é importante verificar que são usados métodos seguros, capazes de garantir a proteção da integridade e da privacidade dos dados trocados. Visto que o *chatbot* desenvolvido para este projeto está *hospedado* na *cloud* e comunica através da *internet*, é necessário ter em conta as ligações seguras de comunicação. Para isto, o *bot* conta as capacidades da *Direct Line* API (ver secção 5.2), que permite comunicação através de HTTPS e que garante a autenticação dos pedidos feitos através de chaves secretas e *tokens*. Já o pacote *npm* [75] utilizado para desenvolver o código de acesso à base de dados garante que a comunicação é feita através de SSL, permitindo assim uma ligação encriptada para a troca de dados.

Capítulo 6
Implementação

Neste capítulo aprofundam-se aspetos relacionados com o desenvolvimento dos vários componentes da solução, dando ênfase aos detalhes da implementação propriamente dita. São também descritas as funcionalidades desenvolvidas para o *chatbot*, de forma a dar resposta aos requisitos funcionais, assim como os testes efetuados.

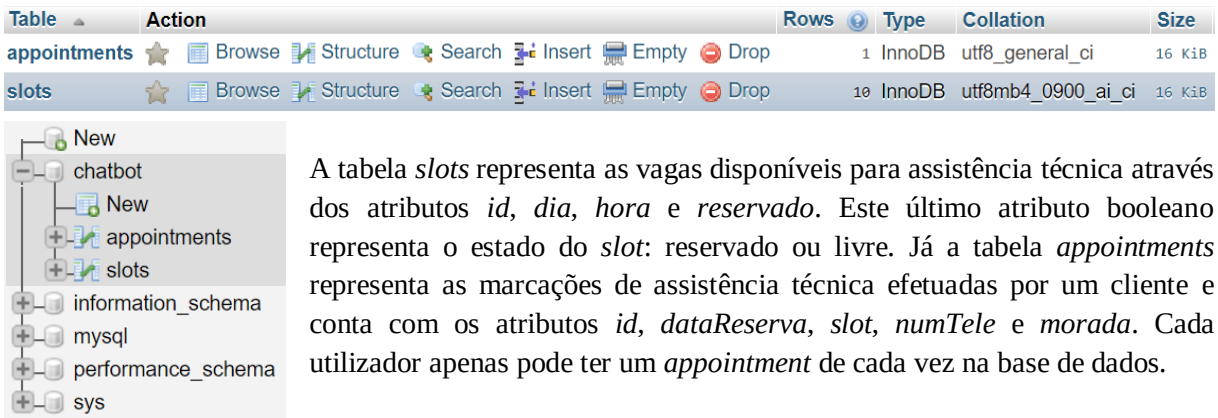
6.1 Chatbot

6.2 Base de dados

Nesta secção será explicado de que forma foi implementado o acesso à base de dados e as decisões associadas ao mesmo. O armazenamento numa base de dados foi considerado devido à existência de requisitos funcionais que necessitam de persistência para ser devidamente concretizados (RF3 e RF4). Para isto, foi usado o *MySQL* para permitir armazenamento relacional. Este abordagem foi escolhida devido à natureza do projeto, em que apenas é necessário guardar tipos de dados simples e estruturados, e onde os requisitos são claros e conhecidos. Outros fatores que contribuíram para esta decisão foram a facilidade de integração do SQL em sistemas já existentes, assim como a grande oferta de bibliotecas, suporte outras ferramentas. Apesar da abordagem não-relacional (*NoSQL*) ter sido considerada devido ao seu bom desempenho e escalabilidade, foi decidido que o seu uso não era justificado no contexto do presente projeto. Como mencionado na secção 2.7, as bases de dados *NoSQL* são indicadas para armazenar dados complexos e não-estruturados, não sendo a escolha ideal para dados simples como os que são armazenados no caso do *chatbot* desenvolvido.

Através da interface gráfica *phpMyAdmin*, uma ferramenta *open-source* de administração *MySQL*, foi criada uma base de dados com o nome *chatbot*. De seguida, através de *queries SQL* à base de dados, foram criadas duas tabelas: *appointments* e *slots*.

***Figura 6.2** - Representação da base dados na interface phpMyAdmin*



A tabela *slots* representa as vagas disponíveis para assistência técnica através dos atributos *id*, *dia*, *hora* e *reservado*. Este último atributo booleano representa o estado do *slot*: reservado ou livre. Já a tabela *appointments* representa as marcações de assistência técnica efetuadas por um cliente e conta com os atributos *id*, *dataReserva*, *slot*, *numTele* e *morada*. Cada utilizador apenas pode ter um *appointment* de cada vez na base de dados.

Como mencionado na secção 6.1, a linguagem de programação utilizada para desenvolver o *bot* foi o *TypeScript*, uma extensão do *JavaScript*. Visto que o ambiente de desenvolvimento utilizado é o *Node.js*, o acesso à base de dados foi feito através do módulo *Node MySQL2* [75]. Este cliente *MySQL* oferece bom desempenho e suporta diversas funcionalidades como *prepared statements*, compressão e SSL. Depois de proceder à instalação do mesmo no diretório do projeto através do *npm*, o módulo foi importado nas *skills* que necessitam de acesso à base de dados, sendo estas *SkillAgendamento* e *SkillCancel*. De seguida, foi necessário invocar o método *createConnection()* e preenchê-lo com os parâmetros que permitem efetuar a ligação à base de dados criada anteriormente, como o *hostname*, o *username* e a *password*. Após estabelecer a conexão, e dependendo da intenção do utilizador e da informação que fornece ao *bot*, são feitos *selects*, *inserts* ou *updates* de forma a consultar ou a inserir dados nas tabelas. O módulo *MySQL2* permite utilizar *promises* como *wrappers*, permitindo efetuar estes pedidos à base de dados de forma mais compacta como mostrado no exemplo seguinte:

```
async function deleteAppointment(): Promise<void> {
  try {
    const query = 'DELETE FROM appointments WHERE dataReserva = ? AND slot = ?'
    await connection.execute(query, [dataAppoint, horaAppoint])
  } catch (e) {
    logger.info(e)
  }
}
```

A *query* anterior é um exemplo claro da forma como o *MySQL2* ajuda a prevenir injeções SQL, como referido na secção 5.3. Caso as variáveis *dataAppoint* e *horaAppoint* fossem referenciadas dentro da variável *query*, alterações maliciosas ao valor das mesmas poderiam comprometer a segurança da base de dados. A utilização de '?' como *placeholder* garante o *escaping* dos dados fornecidos pelo utilizador, neste caso as variáveis globais *dataAppoint* e *horaAppoint*.

6.3 Interface gráfica

Nesta secção será descrito de que forma foi integrada e customizada a interface gráfica que permite a interação dos utilizadores com o *chatbot*. Como mencionado anteriormente, para este efeito foi utilizado o *Web Chat* da *BotFramework*, um componente de *chat* altamente customizável e *open-source*. Este componente foi posteriormente embutido numa página *web*, como será explicado nesta secção.

Apesar de idealmente ser suposto integrar o *chatbot* desenvolvido numa aplicação pré-existente (exemplo: *App* Cliente NOS), visto estarmos perante um protótipo cujo objetivo é ser demonstrado a possíveis clientes, foi seguida uma abordagem móvel *web*. A mesma garante que o *chatbot* se adapte a qualquer tipo de dispositivo ou sistema operativo, bastando apenas ter um *web browser* instalado no dispositivo móvel. Apesar de ter sido considerada a abordagem híbrida, não foram identificadas vantagens significativas que justificassem o seu uso, já que a solução móvel desenvolvida não está fortemente dependente de acesso a *hardware* do dispositivo nem de capacidades de desempenho nativas.

Para tornar o *bot* acessível através do *Web Chat* executaram-se aos seguintes passos:

1. Criação de um ficheiro *index.html*
   Este ficheiro, que conta com o código relacionado com o *front-end* e com a síntese de voz, foi criado de modo a poder ser hospedado no *GitHub Pages*, um serviço de *hosting* que funciona a partir dos repositórios *GitHub*. Desta forma, é possível aceder ao *Web Chat* e ao *bot* através de qualquer *browser*.

2. Importação do *script* do *Web Chat*
   Já dentro de *index.html*, procedeu-se à importação do *script* da CDN (*Content Delivery Network*) que dá acesso ao *Web Chat*. Isto é feito através da inclusão da linha *<script src="https://cdn.botframework.com/botframework-webchat/latest/webchat.js"></script>* no corpo do ficheiro *html*. O uso de uma CDN traz vários benefícios como melhor tempo de carregamento do *website*, menores custos de largura de banda, maior disponibilidade e maior segurança [82].

3. Renderização do *Web Chat*
   Para permitir a renderização do *Web Chat* foi adicionado ao corpo do ficheiro uma função assíncrona. Esta conta com a chamada ao método *renderWebChat*, onde é possível definir vários atributos. O atributo chave para a renderização é o *directLine*, preenchido com *token* de acesso à *Direct Line* API, obtido a partir da plataforma *Azure*. Este *token* torna possível a comunicação entre o *bot* e o *Web Chat*.

4. Customização do *Web Chat*
   De modo a poder customizar a aparência conforme desejado foi necessário criar um objeto *styleOptions*, preenchê-lo e passá-lo ao *Web Chat*. Esta passagem é também feita dentro do método *renderWebChat*, mencionado anteriormente. O objeto *styleOptions* foi preenchido com diversos atributos de forma a mudar o aspeto da interface de conversação. O resultado obtido pode ser observado na Figura 6.3.

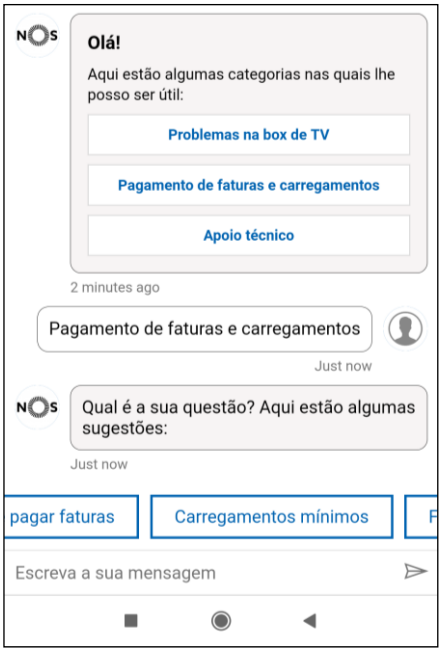


***Figura 6.3** – Interface gráfica do chatbot*

6.4 Síntese de voz

De seguida será explicado de que forma foi feita a integração de voz no *chatbot*, procurando satisfazer o RF6. Sendo a interação entre o utilizador e o *bot* feita através de um *web browser*, tirou-se partido das capacidades da *Web Speech* API [77] para a implementação da síntese de voz. No entanto, de modo a poder acomodar as customizações necessárias ao contexto do projeto, foi preciso recorrer também a um *middleware*. O *Redux* [79], o *middleware* usado, permite intercetar as mensagens trocadas entre o *bot* criado e o *Web Chat*, fazendo com que seja possível alterar certos tipos de comportamentos da aplicação. O *middleware* conta com um objeto *store*, onde é guardado o estado da aplicação. É através desta *store* que se torna possível intercetar vários tipos de ações e atividades enviadas pelo *bot*.

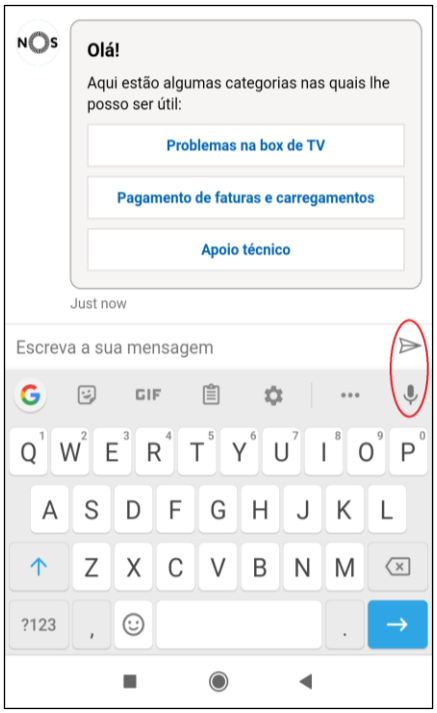


***Figura 6.4** – Representação dos botões "send" e "microphone"*

No ficheiro *index.html* e a partir do método *createStore*, foi criado o objeto *store*. Através do *middleware*, este objeto foi expandido conforme necessário e posteriormente passado ao *Web Chat*. É dentro deste objeto que se encontra o código relacionado com a síntese de voz e onde são endereçados vários problemas relacionados com a mesma. Alguns exemplos são a passagem para voz portuguesa, a capacidade de ativar e desativar a voz, a leitura de *hero cards*, a omissão de URL's adjacentes a hiperligações e problemas relacionados com a pronúncia de certas palavras. Os problemas referidos são explicados na secção 7.2.

Apesar da *Web Speech* API também permitir incorporar *speech-to-text* através da *SpeechRecognitionInterface*, foi decidido utilizar o reconhecimento de voz presente por omissão nos dispositivos móveis (*Speech Recognizer*, no *Android* e a *Speech Framework* no *iOS*). Esta escolha permite ao utilizador alternar entre *input* manual de texto e voz conforme necessário, como mostrado na Figura 6.4.

6.5 Funcionalidades

De forma a satisfazer os requisitos funcionais apresentados em 4.1.1, foram desenvolvidas para o *chatbot* as funcionalidades descritas nesta secção. Por razões especificadas na secção 7.2, as funcionalidades serão mostradas em interfaces diferentes, sendo elas o terminal e o *webchat* num dispositivo móvel.

<u>Resolução de problemas na box de TV</u>:

O *bot* está preparado para responder a várias perguntas relacionadas com problemas comuns que possam surgir numa box de TV da NOS. Cada um destes problemas tem um *intent* correspondente, que será reconhecido caso o *input* do utilizador contenha certas palavras ou combinações das mesmas. Apesar da NOS disponibilizar diferentes tipos de boxes [76], as sugestões apresentadas pelo *bot* são legítimas para o conjunto dos diferentes modelos disponibilizados. Luzes desligadas, formatos errados de imagem ou a não-existência de som são exemplos de problemas que se enquadram no âmbito deste esclarecimento. As *intents* relacionadas com esta funcionalidade têm um nome na forma *Ajuda{Assunto}*, no qual *Assunto* representa o tipo de problema a ser endereçado. Estas *intents* são mapeadas aos diálogos correspondentes através da *skill Faq*, que apenas devolve uma mensagem. A mensagem presente no diálogo é então retornada ao utilizador. Na figura seguinte podemos ver dois exemplos simples de interações deste tipo:

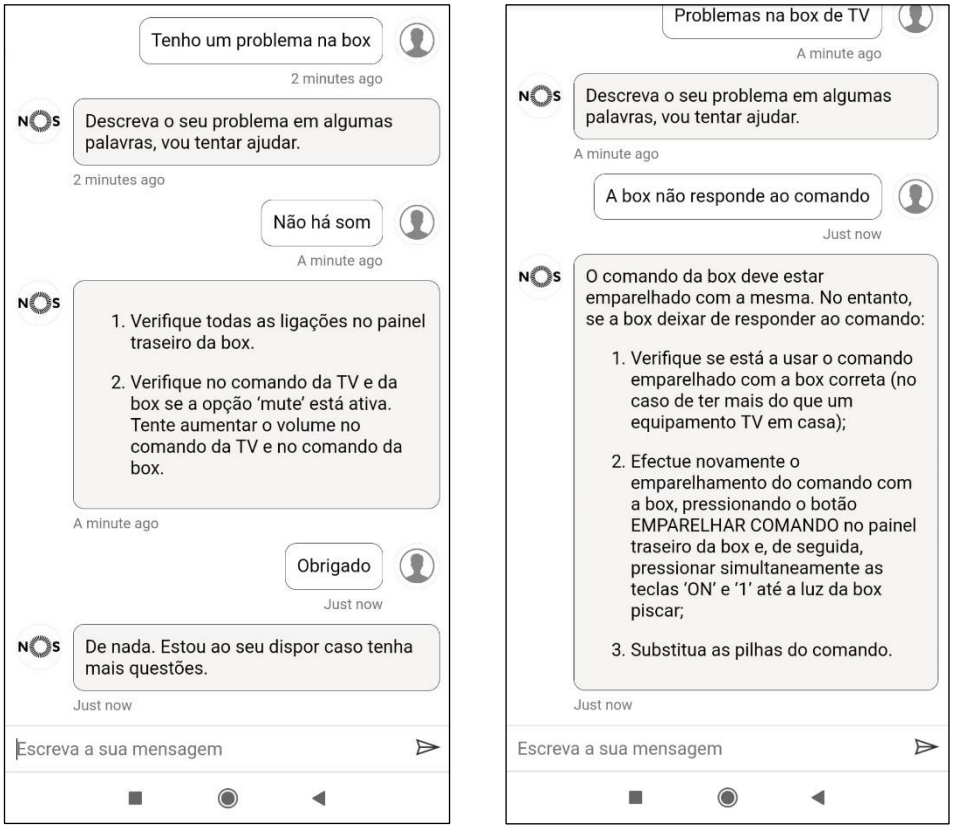


Figura 6.5 - *Interação com o chatbot (Resolução de problemas)*

<u>Dúvidas sobre pagamentos e carregamentos</u>:

Para além de perguntas relacionadas com as boxes da NOS, o *bot* também está preparado para responder a questões relacionadas com pagamentos de faturas e carregamentos de telemóvel. Desta funcionalidade fazem parte questões sobre as diferentes formas de carregamento disponíveis, quais os valores mínimos de carregamento ou quais os modos de pagamento de fatura disponíveis para o cliente. As *intents* relacionadas com esta funcionalidade têm normalmente no nome a palavra *Carregamento* ou *Pagamento*. Assim como na funcionalidade anterior, é através da *skill Faq* que as mensagens de resposta apropriadas são retornadas para os utilizadores. Na figura seguinte podemos observar dois exemplos simples de interações deste tipo:

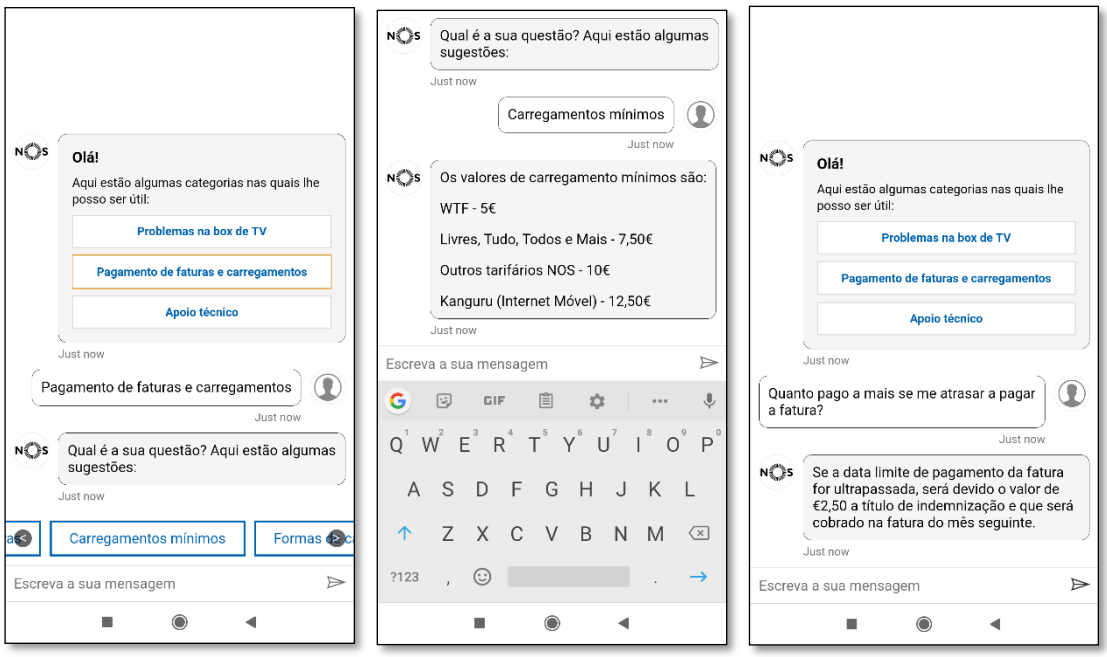


***Figura 6.6** – Interação com o chatbot (Faturas e carregamentos)*

<u>Agendamento de assistência técnica</u>:

Esta funcionalidade pretende simular o agendamento de assistência técnica na casa do cliente, idealmente em casos em que este não consiga resolver os problemas com que se deparou através das sugestões dadas pelo *chatbot*. Este processo é feito com recurso a uma base de dados local *MySQL* (ver secção 6.2), na qual são persistidos os agendamentos e geridos os *slots* disponíveis. Caso ainda não tenha um agendamento marcado, o utilizador pode pedir ao *bot* para iniciar o processo, fornecendo os dados necessários e escolhendo um horário no qual haja técnicos disponíveis. O utilizador pode ainda retroceder e corrigir os dados, conforme necessário. O processo de agendamento é feito com recurso a um *intent* e a um *dialog*, ambos com o nome *Agendamento*. Os mesmos estão associados à *SkillAgendamento*, responsável por pedir ao utilizador a informação necessária, processá-la e persisti-la posteriormente na base de dados. Os dados introduzidos pelo utilizador são validados através de expressões regulares presentes nas *entities*. Já o acesso à base de dados é feito através da biblioteca *MySQL2* [75], um cliente *MySQL* para *Node.js* focado no desempenho e que conta com funcionalidades

como compressão, SSL e *prepared statements*. Nas figuras seguintes podemos observar um exemplo de uma interação deste tipo:

```
[INFO] 16:11:36 AdaptorTerminal: emulating( Bot Tese [development] )
Message: Agendar assistência em casa
[INFO] 16:11:44 @ bot send typing on channel: terminal
[INFO] 16:11:44 @ bot send { Text: 'Para que dia deseja efetuar a marcação?' }
Message: 10/06/2019
[INFO] 16:11:50 @ bot send typing on channel: terminal
[INFO] 16:11:50 @ bot send { Text:
   'Escolha uma das horas disponíveis para o dia 10/06/2019:  {eol} 1 - 10:30 {eol} 2 - 11:30
{eol} 3 - 17:00 {eol} 4 - 15:30 {eol} ' }
Message: 3
[INFO] 16:11:59 @ bot send typing on channel: terminal
[INFO] 16:11:59 @ bot send { Text: 'Por favor, introduza a sua morada:' }
Message: Rua José Aranha, Nº2, 4ºA, 1320 Lisboa
[INFO] 16:13:10 @ bot send typing on channel: terminal
[INFO] 16:13:10 @ bot send { Text: 'Indique também o seu número de telemóvel:' }
Message: 923325634
[INFO] 16:13:17 @ bot send typing on channel: terminal
[INFO] 16:13:17 @ bot send { Text:
   'Confirmar agendamento de assistência técnica para o dia 10/06/2019 às 17:00? ' }
Message: Sim
[INFO] 16:13:20 @ bot send typing on channel: terminal
[INFO] 16:13:20 @ bot send { Text: 'Agendamento marcado.' }
Message:
```

Figura 6.7 – *Exemplo de um agendamento de assistência técnica*

```
[INFO] 16:23:03 @ bot send typing on channel: terminal
[INFO] 16:23:03 @ bot send { Text:
   'Escolha uma das horas disponíveis para o dia 10/06/2019:  {eol} 1 - 10:30 {eol} 2 - 11:30
{eol} 3 - 17:00 {eol} 4 - 15:30 {eol} ' }
Message: 4
[INFO] 16:23:09 @ bot send typing on channel: terminal
[INFO] 16:23:09 @ bot send { Text: 'Por favor, introduza a sua morada:' }
Message: anterior
[INFO] 16:23:14 @ bot send typing on channel: terminal
[INFO] 16:23:14 @ bot send { Text:
   'Escolha uma das horas disponíveis para o dia 10/06/2019:  {eol} 1 - 10:30 {eol} 2 - 11:30
{eol} 3 - 17:00 {eol} 4 - 15:30 {eol} ' }
Message: 3
[INFO] 16:23:17 @ bot send typing on channel: terminal
[INFO] 16:23:17 @ bot send { Text: 'Por favor, introduza a sua morada:' }
Message: Rua
```

Figura 6.8 – *Exemplo no qual o utilizador se engana na hora escolhida, volta atrás e escolhe outra hora*

```
[INFO] 16:26:05 @ bot send { Text:
   'Escolha uma das horas disponíveis para o dia 10/06/2019:  {eol} 1 - 10:30 {eol} 2 - 11:30
{eol} 3 - 17:00 {eol} 4 - 15:30 {eol} ' }
Message: 3
[INFO] 16:26:08 @ bot send typing on channel: terminal
[INFO] 16:26:08 @ bot send { Text: 'Por favor, introduza a sua morada:' }
Message: cancelar
[INFO] 16:26:15 @ bot send typing on channel: terminal
[INFO] 16:26:15 @ bot send { Text: 'Operação cancelada.' }
Message:
```

Figura 6.9 – *Exemplo no qual o utilizador cancela o processo de agendamento*

Cancelamento/reagendamento de assistência técnica:

Para além de permitir o agendamento de assistência técnica na casa do cliente, também é possível fazer o cancelamento e posterior reagendamento do mesmo. Quando um utilizador comunica a intenção de cancelar ou reagendar uma marcação feita previamente, o *bot* verifica na base de dados se o mesmo já tem um agendamento feito. Caso isto se verifique, é dada ao utilizador a opção de cancelá-lo, apagando o agendamento da base de dados e libertando o horário correspondente. Posteriormente, é perguntado ao utilizador se deseja fazer um novo agendamento. O diálogo e o *intent* correspondente, *CancelAgend*, possibilitam o cancelamento através da *skill* associada aos mesmos, *SkillCancel*, que trata de toda a lógica relativa a esta funcionalidade. Na figura seguinte podemos observar um exemplo de uma interação deste tipo:

```
Message: cancelar assistência técnica
[INFO] 17:28:30 @ bot send typing on channel: terminal
[INFO] 17:28:30 @ bot send { Text:
   'Tem um agendamento marcado para o dia 10/06/2019 às 15:30, na morada Rua das Flores, Nº2,
3º A, 1340 Lisboa. {eol} Deseja cancelá-lo?' }
Message: Sim
[INFO] 17:28:37 @ bot send typing on channel: terminal
[INFO] 17:28:37 @ bot send [ { Title: 'FinishCard',
    Text:
     'O seu agendamento foi cancelado. Pode marcá-lo para outro dia.',
    Buttons: [ 'Reagendar assistência técnica' ] } ]
Message:
```

Figura 6.10 - *Exemplo do processo de cancelamento e reagendamento de assistência técnica*

Download de manuais de utilização:

O *bot* é capaz de apresentar ao utilizador os manuais de utilização de diversos equipamentos da NOS como *boxes* de TV, comandos e descodificadores digitais. Para isto, o utilizador deve comunicar ao *bot* que deseja ver o manual de um determinado equipamento. Como resposta, é apresentada uma ligação que permite acesso direto ao mesmo. De forma a poder devolver a ligação adequada ao utilizador, o *bot* faz uso da *entity Equipamento*. Esta pode assumir diversos valores e deve estar presente no pedido, como ilustrado numa das imagens apresentadas de seguida. O diálogo *DownloadManual* faz uso das *RequiredEntities* (ver secção 2.4) de modo a garantir que uma entidade do tipo *Equipamento* está presente. Já o *intent* associado invoca a *skill SkillShowManual*, que permite devolver o manual adequado de acordo com o tipo de entidade presente. Nas figuras em baixo podemos ver um exemplo de uma interação deste tipo:

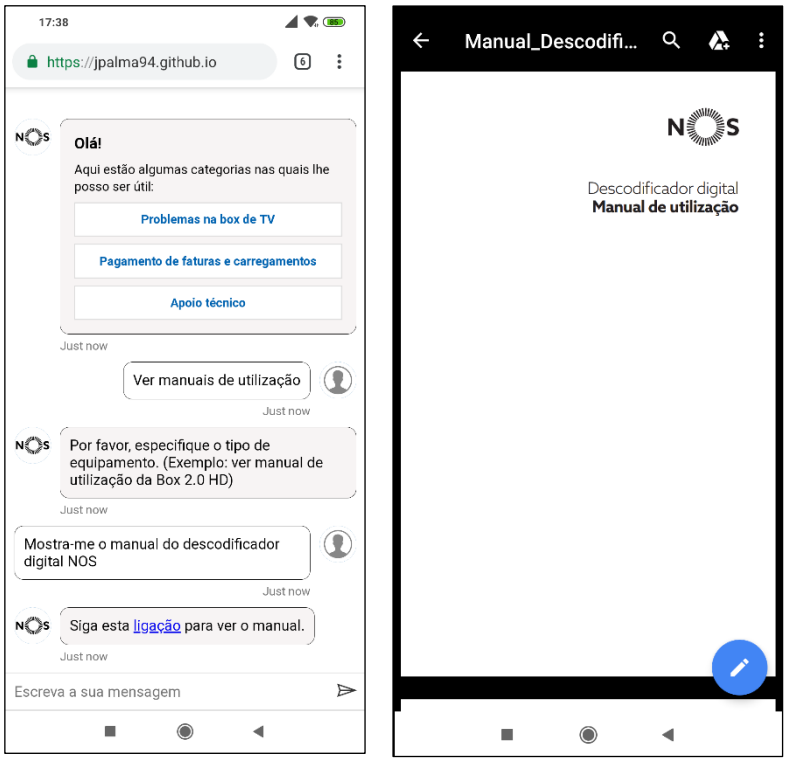


Figura 6.11 *- Interação com o chatbot (Download de manuais de utilização)*

<u>Falar com assistente</u>:

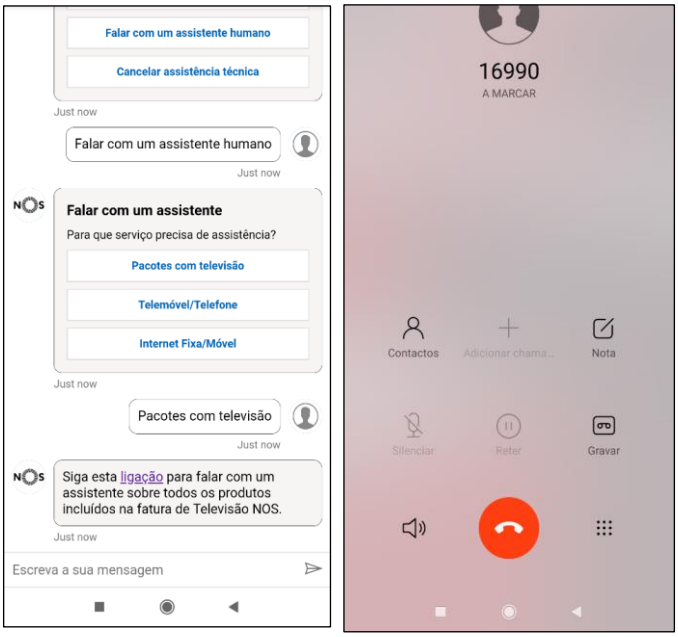


Figura 6.12 *– Interação com o chatbot (Falar com assistente)*

Caso o utilizador não consiga resolver os problemas com que se deparou através das sugestões do *bot*, ou caso simplesmente o deseje, pode optar por falar com um assistente humano. Ao comunicar esta intenção ao *bot*, este fornece-lhe a ligação para a linha de apoio adequada ao tipo de problema em questão. Esta interação é possibilitada pela *intent FalarAssistente* e pelo diálogo com o mesmo nome. Este último, para além do texto que permite fazer a ligação telefónica, contém botões que serão mostrados ao utilizador de modo a guiá-lo na escolha. A *skill* associada ao *intent* é a *SkillChamada*, responsável por retornar a mensagem adequada ao utilizador de acordo com o tipo de entidade reconhecida. Nas imagens seguintes pode observar-se o exemplo de uma interação deste tipo:

Capacidade de comunicar através de voz:

O *bot* desenvolvido conta com a capacidade de fazer síntese e reconhecimento de voz, de forma a facilitar a interação em alguns dos casos. Por omissão, desde o início da interação entre o utilizador e o *bot*, as mensagens de texto enviadas por este último são sintetizadas e faladas em linguagem natural. A voz que resulta da síntese é feminina e a linguagem usada é o português, com pronúncia de Portugal. A presença de voz é possível devido à *Web Speech API* [77]. O uso da mesma, em conjunto com o *web chat* da *Microsoft* [78] e com o *middleware Redux* [79] são explicados e detalhados na secção 6.4. Apesar de ser possível implementar reconhecimento de voz através da *Web Speech* API, dada a natureza móvel do projeto a desenvolver, optou-se por utilizar o serviço de reconhecimento de voz presente por omissão nos dispositivos móveis (*Speech Recognizer*, no *Android* e a *Speech Framework* no *iOS*). Através da implementação de um diálogo e de um *intent* para o efeito, é possível ao utilizador ativar ou desativar a voz do *bot* de forma rápida. Na figura seguinte pode observar-me um exemplo de uma interação deste tipo:

Figura 6.13 *- Exemplo de um utilizador a desativar e a ativar a voz do chatbot*

6.6 Testes

No contexto dos *chatbots*, assim como em qualquer outro tipo de *software*, a fase de testes é de grande importância para a garantia da qualidade do produto criado. No entanto, ao contrário de outras aplicações *web* ou móveis onde a aplicação segue um modo predefinido de interação, os *chatbots* correm sem quaisquer restrições. Adicionalmente, as interfaces conversacionais são complexas e compostas por vários módulos, havendo uma grande diversidade de parâmetros de interação e de dinâmicas temporais. Outro aspeto relevante é que a avaliação de *chatbots* está muito dependente do contexto da aplicação e do objetivo da mesma. Os *bots* orientados a tarefas, por exemplo, têm objetivos específicos que guiam a sua interação. Já os *bots* não orientados a tarefas não têm objetivo, pretendendo apenas estabelecer conversas livres com os utilizadores. Estes fatores fazem com que a forma de testar os mesmos seja diferente, em certos aspetos, da forma de testar *software* tradicional.

6.6.1 Chatbottest

O *Chatbottest* [84] é um guia *open-source* que permite testar o *design* de um *chatbot*. Este guia completo de instruções e perguntas é baseado numa avaliação heurística e na vasta experiência dos seus criadores. Sendo um dos poucos projetos que procuram contribuir estandardização dos testes deste tipo, o conceito do *chatbottest* segue uma abordagem Gaussiana. O mecanismo de teste usado pode ser separado em três categorias distintas: cenários esperados, cenários possíveis e cenários quase impossíveis. Estes cenários podem ser mapeados a distâncias sigma, representadas na figura seguinte:

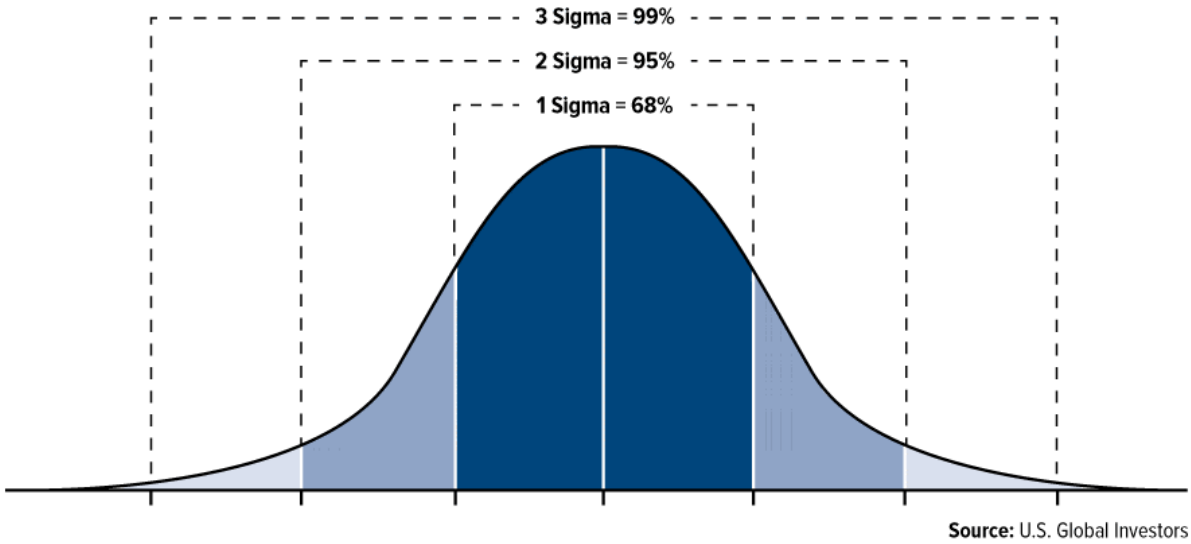


***Figura 6.14** – Representação do desvio padrão (Sigma), que mede o grau de variação da média*

Depois de testar os cenários quase impossíveis, que podem neste caso ser considerados como a distância 3-sigma, o desempenho do *chatbot* seria observado com um intervalo de confiança de 99%. Testar para além deste intervalo não faz sentido, visto que a capacidade humana de produzir *input* para um *bot* não tem limites. Este método de teste, que será aplicado de seguida, foi escolhido pois enquadra-se no contexto do presente projeto. No entanto, a versão usada será ligeiramente adaptada, visto existir um número reduzido de perguntas que não fazem sentido neste contexto. As questões estarão divididas em sete categorias distintas, relativas ao desenho do *chatbot*. As respostas dadas permitirão refletir sobre o estado do que foi desenvolvido até agora, assim como tirar algumas orientações sobre como melhorar o *design* do *bot* criado em iterações futuras.

1 | **Respostas**
(Quais os elementos fornecidos pelo chatbot e de que forma o faz?)

Instruções: Rever todas as mensagens enviadas pelo *bot*.
Questões:

- Envia mais do que uma mensagem de seguida? **Não.**
- Na maioria dos casos, é possível ler a mensagem enviada pelo *bot* sem fazer *scroll*? **Sim.**
- As mensagens estão bem escritas? Há erros ortográficos e/ou de gramática? **Sim. Não.**

Instruções: Repetir a mesma mensagem ao *chatbot* 4 vezes seguidas.
Questão:

- Existem respostas diferentes para a mesma frase? **Não.**

Instruções: Contar o número de caracteres de uma mensagem de tamanho comum.
Questões:

- Quantos caracteres tem a mensagem? **174.**
- Há alguma mensagem com mais de 220 caracteres? **Sim.**

Outras questões:

- O *bot* usa apenas texto simples ou também envia outros elementos como emojis, gifs ou vídeos? **Não.**
- O *bot* usa elementos de *rich media* como botões, *carrousels*, menus persistentes ou *quick reply*? **Sim.**
- O uso dos elementos mencionados na pergunta anterior é feito de forma consistente e nas mesmas circunstâncias ao longo da conversa? **Sim, na maior parte dos casos.**
- Acha o uso dos elementos mencionados justificável ou pode haver soluções mais simples? **Acho justificável.**

2 | **Gestão de erros**
(De que forma lida o chatbot com possíveis erros?)

Instruções: Escrever "ajuda".
Questões:

- Ao responder, o *bot* relembra o seu propósito? **Sim.**
- Ao responder, o *bot* dá opções de como interagir? **Sim.**

Instruções: Escrever algo que o *bot* não entenda, esperar pela mensagem de erro e continuar a dizer o mesmo até à mesma mensagem de erro aparecer.
Questão:

- Há mensagens de erro diferentes para a mesma frase? **Não.**

Instruções: Começar um fluxo de conversação: responder à primeira questão com uma palavra inesperada, esperar pela mensagem de erro e dar uma resposta correta. Repetir isto nos próximos passos da conversa.
Questões:

- As mensagens de erro são relacionadas com a conversa corrente? **Em certos casos.**
- Oferecem uma solução proativa? **Não.**
- O *bot* relembra qual o seu propósito? **Sim.**

Instruções: Tentar falar com um ser humano real através do *bot*.
Questão:

- Há possibilidade de contactar ou falar com pessoas reais? **Sim.**

Questão:

- Estando o *bot* ligado a serviços externos, caso estes falhem, o *bot* informa o utilizador do problema específico? **Não.**

3 | **Inteligência**
(O chatbot possui algum tipo de inteligência?)

Instruções: Fazer um pedido, dentro do âmbito do *chatbot*, onde uma palavra-chave fica de fora de modo a que o pedido seja percebido apenas dentro de um certo contexto.
Questão:

- O *bot* percebe as perguntas? **Sim.**

Instruções: Pedir a alguém conhecido que faça as mesmas questões ao *bot*, mas noutra cidade.
Questões:

- O *bot* tem em conta as localizações diferentes? **Não.**
- O *bot* sabe informação sobre o utilizador e usa-a adaptar as mensagens? **Sim, mas apenas numa das funcionalidades.**
- O *bot* envia sugestões diferentes dependendo do utilizador? **Não.**

Instruções: Depois de uma questão feita pelo *bot*, responder. Reiniciar o *bot* e reproduzir a mesma conversa, mas agora, após a questão do *bot*, esperar 30 minutos para dar uma resposta.
Questões:

- O fluxo da conversa continuou? **Sim.**
- O *bot* fez alguma referência à diferença de tempo? **Não.**

Questão

- O *bot* sabe o nome do utilizador com quem está a falar? **Sim.**

Instruções: Rever as conversas passadas tidas com o *bot* e listar todas as questões feitas pelo mesmo sobre informações pessoais do utilizador.
Questão:

- O *bot* usa a informação dada pelo utilizador para personalizar mensagens de resposta? **Sim, mas apenas numa das funcionalidades.**

4 | **Navegação**
(Quão fácil é navegar na conversa com o chatbot e nos seus diferentes fluxos?)

Instruções: Passar algum tempo a comunicar com o *bot* e seguir os diferentes fluxos de conversação propostos pelo mesmo. Tentar identidicá-los.
Questões:

- Quantos fluxos diferentes foram identificados? **6.**
- Quantos passos há, em média, para obter algo de valor do *bot*? **3.**

Instruções: Começar um fluxo de conversa com mais de 3 passos. Depois de responder a uma questão do *bot*, tentar voltar atrás e mudar a resposta.
Questão:

- O *bot* permite fazê-lo ou é necessário recomeçar o fluxo? **Permite.**

Instruções: Começar um fluxo de conversação e mudar para outro.
Questão:

- O *bot* permite fazê-lo e informa sobre sobre a mudança? **Permite mas não informa de forma clara.**

Instruções: Começar um fluxo de conversa com mais de 3 passos.
Questão:

- O *bot* informa o utilizador em que parte do processo se encontra? **Em algumas funcionalidades.**

Instruções: Rever as mensagens enviados pelo *bot*.
Questão:

- Nas respostas que dá, o *bot* faz referência às perguntas feitas pelo utilizador? **Em algumas.**

5 | **Ambientação**
(Quão fácil é para os utilizadores perceberem o foco do chatbot e de que forma devem interagir com o mesmo?)

Instruções: Iniciar uma conversa com o *bot*.
Questões:

- O *bot* apresenta-se ao utilizador? **Sim.**
- O *bot* explica o seu propóstio? **Sim.**
- O *bot* dá dicas de como interagir? **Sim.**
- Estas dicas são acompanhadas de exemplos, ou do que escrever especificamente? **Não.**

- O *bot* explica cada funcionalidade que suporta desde o início? **Não.**

6 | **Personalidade**
(O chatbot tem uma personalidade clara, adequada aos utilizadores e à conversação em curso?)

Questões:
- O *bot* tem um nome fácil de lembrar? **Não.**
- O *bot* tem uma foto de perfil? **Sim.**
- O *bot* tem uma voz e um tom de voz que são consistentes com a conversa? **Sim.**

Instruções: Refletir sobre o propósito do *bot*.
Questão:
- A voz e o tom encaixam no propósito do *bot*? **Sim.**

Instruções: Refletir sobre o tipo de utilizador do *bot*.
Questão:
- A voz e o tom são adequados a este tipo de utilizador? **Sim.**

Instruções: Forçar uma situação constrangedora, como mostrar frustração ou dizer palavrões.
Questão:
- O *bot* relaxa o tom de voz para tentar acalmar a situação? **Não.**

7 | **Compreensão**
(Quais os elementos do input do utilizador que o chatbot é capaz de perceber e processar?)

Instruções: Enviar duas mensagens seguidas ao *bot* antes deste dar uma resposta.
Questão:
- O *bot* responde a ambas as mensagens? **Sim.**

Instruções: Em vez de responder com as respostas propostas, tentar responder de outra forma mas com o mesmo significado.
Questão:
- O *bot* percebe a resposta? **Em alguns casos.**

Instruções: Forçar uma situação constrangedora, como mostrar frustração ou dizer palavrões.
Questões:
- O *bot* dá respostas diferentes de acordo com o tipo de insulto? **Não.**
- O *bot* tenta acalmar o utilizador e resolver a situação? **Parcialmente.**

Instruções: Dizer algo simpático ao *bot*, elogiá-lo.
Questão:
- O *bot* reage a estas situações? **Não.**

Instruções: Dizer "olá", "adeus" ou "obrigado" ao *bot*.
Questão:
- O *bot* reage adequadamente à situação? **Sim.**

Instruções: Há instruções como “menu”, “pára” ou “ajuda”, que já se começam a tornar instruções padrão.
Questão:

- O *bot* percebe-as? **Sim.**

Tabela 6.1 *- Resultados do Chatbottest*

	Número de questões	Respostas favoráveis	Respostas parcialmente favoráveis	Respostas infavoráveis	Resultado (%)
1 - Respostas	10	6	2	2	70
2 - Gestão de erros	8	4	1	3	56
3 - Inteligência	8	3	2	3	50
4 - Navegação	6	2	4	-	66
5 - Ambientação	5	3	-	2	60
6 - Personalidade	6	4	-	2	66
7 - Compreensão	7	3	2	2	57
TOTAL	50	25	11	14	-

Dado o facto de algumas respostas não serem totalmente favoráveis nem totalmente infavoráveis, foi necessário incluir uma coluna para as respostas parcialmente favoráveis. Exemplos de respostas deste tipo são as que se verificam, mas só em alguns casos. Para o resultado de cada categoria, metade das respostas parcialmente favoráveis são contadas como favoráveis enquanto a outra metade é contada como infavorável. Os resultados são aproximações que indicam os aspetos mais fortes e fracos do *chatbot* desenvolvido, permitindo identificar as categorias que necessitam de mais atenção e melhorias. Como se pode observar na tabela 6.1, existem diversas categorias que precisam de ser melhoradas de forma a atingir o padrão de qualidade desejado para um *chatbot* ideal.

6.6.2 Testes de usabilidade

Os testes de usabilidade consistem numa técnica que permite avaliar um produto de *software* através da interação do mesmo com os seus utilizadores alvo. No caso deste projeto, os testes de usabilidade serão realizados por voluntários que terão de efetuar diversas tarefas ao interagir com o *chatbot* desenvolvido. Estas tarefas (ver Tabela 6.2) estão diretamente relacionadas com as funcionalidades e com os requisitos funcionais explorados em capítulos anteriores. Estes testes contaram com a participação de 7 voluntários, com idades compreendidas entre os 21 e os 63 anos, sendo 71% do sexo masculino e 29% do feminino.

Tabela 6.2 *- Tarefas propostas aos voluntários*

Tarefa	Descrição
1	Pôr uma questão sobre um problema na box de TV (Exemplos: a box não liga, a box não responde ao comando, a imagem está distorcida, não há som...)
2	Saber quais as diferentes formas de pagamento de uma fatura
3	Saber quais os valores mínimos de carregamento do telemóvel
4	Agendar assistência técnica para o dia 10 de Agosto
5	Cancelar a marcação de assistência técnica efetuada previamente
6	Ver o manual de utilização de um equipamento (Exemplos: Box 1.0 HD, Box 2.0 HD, Descodificador digital, Comando Íris, Comando UMA...)
7	Estabelecer contacto com um assistente humano

Tabela 6.3 *- Cotação da dificuldade sentida a realizar uma tarefa*

Dificuldade					
Muito fácil	Fácil	Médio	Difícil	Muito Difícil	Impossível
5	**4**	**3**	**2**	**1**	**0**

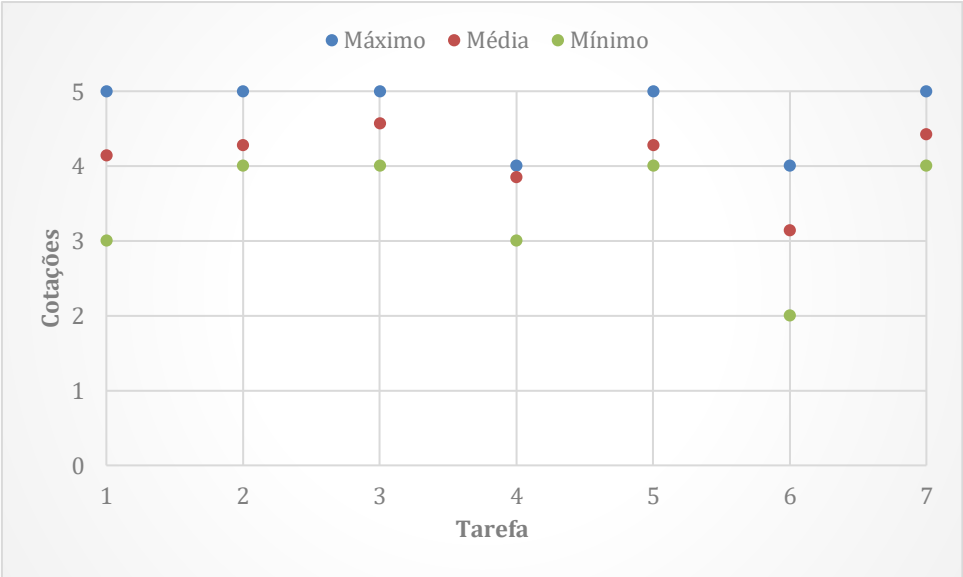


Figura 6.15 *– Gráfico das cotações máximas, médias e mínimas para cada tarefa*

A partir do *feedback* obtido através das respostas dos voluntários foi produzido o gráfico da Figura 6.15, no qual foram representadas as cotações mínimas, médias e máximas de cada tarefa. Ao observar este gráfico, verifica-se que todas as funcionalidades necessitam de algumas melhorias. As tarefas 2,3,5 e 7 foram consideradas acessíveis e fáceis, apesar de ainda deixarem espaço para algumas melhorias. Já no cumprimento das tarefas 1 e 4 verificou-se alguma dificuldade por parte de alguns voluntários, o que sugere que as mesmas necessitam de mais atenção. A tarefa 6, por sua vez, não foi considerada fácil nem intuitiva pela maioria dos voluntários, mostrando que necessita de várias alterações, ou até mesmo de uma reformulação. De seguida, serão mencionadas algumas sugestões de alterações ao *chatbot*, obtidas a partir do *feedback* dos utilizadores:

- Reconhecer as duas variantes da escrita portuguesa: com e sem o acordo ortográfico;
- Reconhecer palavras importantes quando estas têm erros, sugerindo a forma correta de a escrever;
- Obter *feedback* mais regular do utilizador, de forma a ter a informação de se está no bom caminho ou não;
- Tornar mais explícito para os utilizadores quais os tipos de equipamento suportados;
- Permitir de imediato o reagendamento de assistência técnica, acabando com a necessidade de cancelar o mesmo;
- Ter uma opção na mensagem de boas-vindas relativa ao *download* de manuais de equipamentos;
- Adicionar *small talk* (discurso informal) de forma a conferir mais personalidade.

Capítulo 7
Considerações Finais

Este capítulo final apresenta uma visão geral sobre o projeto realizado ao longo do estágio. Para além de um resumo do trabalho realizado, será feita uma reflexão crítica sobre o mesmo, mencionando os principais obstáculos encontrados. Será ainda feita uma análise sobre o possível trabalho a desenvolver no futuro.

7.1 Trabalho realizado

Este trabalho, realizado no contexto da disciplina PEI, teve como objetivo desenvolvimento de uma solução móvel para a vertente B2C. A solução desenvolvida, um *chatbot* assistido por voz, procurou aliar os benefícios da interação através de voz às capacidades de otimização oferecidas pelos *chatbots* no contexto do apoio ao cliente. Desenvolvido num contexto empresarial, o protótipo concebido teve como objetivo final poder ser demonstrado a diversos clientes da *Accenture* na área das telecomunicações. A participação neste projeto proporcionou ao estagiário conhecimentos tanto a nível teórico como prático, permitindo ao mesmo ter contacto direto com diversas metodologias e técnicas utilizadas para o desenvolvimento de *chatbots*.

O trabalho desenvolvido começou com uma análise detalhada e comparativa dos vários conceitos e abordagens relativos ao estado da arte. Foram também explorados casos de estudo específicos e historicamente relevantes no contexto dos *chatbots* e dos assistentes virtuais. Depois de definir os aspetos relacionados com o planeamento e com o levantamento de requisitos, procedeu-se ao desenho da solução e à respetiva implementação. A arquitetura do sistema concebida na fase inicial do estágio manteve-se relativamente fiel à arquitetura final, apesar das várias alterações que ocorreram ao longo do tempo. Já a fase de implementação esteve sujeita a várias mudanças e incertezas, principalmente devido ao atraso no acesso à *framework* da *Accenture* e aos períodos de aprendizagem e de adaptação exigidos pela mesma. Apesar destes imprevistos, a *framework* permitiu o desenvolvimento de um *bot* com funcionalidades variadas, alinhado com os requisitos definidos inicialmente. O mesmo começou por estar disponível apenas localmente, acedendo a uma base de dados *MySQL* mas sem qualquer interface gráfica ou suporte de voz. Para endereçar estas questões foi feito *deploy* do *bot* para a *cloud* da *Microsoft*, o que tornou possível a integração do mesmo com o componente de *chat* utilizado como interface gráfica e com a API de reconhecimento e de síntese de voz. Tanto o *front-end* da solução como a integração de voz necessitaram de alguns ajustes de forma a atingir o resultado pretendido. A interface gráfica foi customizada de modo a ser mais intuitiva e adequada a uma situação de apoio ao cliente. No caso da síntese de voz, foi necessário recorrer a *middleware* para alterar o alguns tipos de comportamento, nomeadamente a pronúncia de certas palavras de língua portuguesa e a omissão de URL's adjacentes a hiperligações. Numa fase final, foram efetuados testes de forma a verificar e a validar as características do *software* desenvolvido. Os mesmos permitiram um aprofundamento da percepção do estagiário sobre quais os aspetos mais fortes e quais os aspetos a melhorar em iterações futuras.

7.2 Dificuldades encontradas

As dificuldades mais significativas encontradas no decorrer do projeto podem dividir-se em dois tipos: dificuldades técnicas e dificuldades organizacionais. A primeira dificuldade técnica prende-se com o processo de aprendizagem e de adaptação à *framework* utilizada, interna à *Accenture*. Tanto a documentação limitada como a não-existência de qualquer recurso de apoio *online* foram fatores que contribuíram para que este processo fosse mais demorado do que o suposto. A segunda dificuldade técnica esteve relacionada com os ajustes necessários à síntese de voz. A pronúncia de certas palavras e expressões (exemplos: *NOS*, *phone*, etc..) teve de ser ajustada para se adaptar à voz portuguesa usada no *chatbot*. Já as várias mensagens que continham hiperligações tiveram também de ser intercetadas e adaptadas de forma a evitar que o *bot* proferisse os URL's associados.

A dificuldade técnica mais significativa esteve relacionada com o acesso à base de dados *MySQL*. Apesar das funcionalidades que necessitam de acesso à base de dados terem sido completadas com sucesso, as mesmas apenas podem ser concretizadas ao correr o *bot* localmente. Após ter sido feito o *deploy* para a *cloud* da *Microsoft*, a comunicação entre o *bot* e a base de dados deixou de ser possível. O estagiário deparou-se com um erro EACCES (*"Permission denied"*) sempre que o *bot* pretendia aceder ao porto 3306 (destinado a conexões *MySQL*). Apesar de existirem alternativas para contornar este problema, na etapa do projeto em que o mesmo foi detetado já não seria realista voltar atrás e reescrever toda a lógica de acesso à base de dados. Devido às dificuldades mencionadas, na secção 6.5, as funcionalidades que usam necessitam da base de dados são mostradas localmente, através do terminal. Todas as outras funcionalidades não foram afetadas por esta falta de permissões.

Já a nível das dificuldades organizacionais, é de destacar a dificuldade de acesso do estagiário à *framework* de desenvolvimento (*BotFactory*). Sendo os três responsáveis pela mesma funcionários do pólo de Braga da *Accenture*, o contacto entre estes e o estagiário foi muitas vezes complicado e dependente de horários sobrecarregados. Apesar do esforço para tentar marcar mais reuniões, ocorreu um atraso significativo no acesso à *framework*. Este atraso de quase dois meses impactou o tempo disponível para o desenvolvimento da solução proposta (apesar de ter dado oportunidade ao estagiário de explorar outras plataformas de desenvolvimento, como a *Microsoft Bot Framework*). Já depois do *bot* contar com as funcionalidades principais e de estar funcional localmente, ocorreu outro atraso, desta vez no processo de *deployment*. O estagiário esteve dependente da disponibilidade dos funcionários mencionados anteriormente para ser guiado no processo de *deployment* para a *cloud*, específico à *framework* interna usada. Este atraso de quase um mês resultou na redução do tempo disponível para integrar o *bot* com a interface gráfica e com a voz.

7.3 Trabalho futuro

Estando a tecnologia em constante evolução e atualização, é importante que as soluções de *software* acompanhem estas mudanças. Sendo o *chatbot* desenvolvido um protótipo a apresentar a clientes da *Accenture*, faz sentido considerar atentamente as possibilidades de melhoramento do mesmo. Após a conclusão do estágio na *Accenture* e do presente projeto, sugerem-se os tópicos seguintes como trabalho futuro:

- Expandir a base de conhecimento do *bot*, aumentando o número de perguntas e respostas, contemplando mais tipos de equipamentos diferentes e melhorando as sugestões dadas ao utilizador. Para otimizar este processo, seria interessante criar um *script* para integrar na *framework* que permitisse adicionar novas perguntas e respostas de forma mais rápida.

- Adicionar suporte para outros idiomas, permitindo ao *bot* reconhecer a linguagem usada pelo utilizador e responder de acordo com a mesma.

- Rescrever toda a lógica de acesso à base de dados de forma a que este acesso seja compatível com a presença do *chatbot* no *Azure*, colmatando assim as falhas suscitadas pelo problema de permissões referido na secção 7.2. Isto poderia ser feito através da criação de uma base de dados SQL nos serviços *cloud* do *Azure*, permitindo ao *bot* comunicar sem restrições com a mesma.

- Dando seguimento aos resultados obtidos na secção 6.6, adicionar mais inteligência ao *bot*, permitindo que o mesmo identifique e processe informação relevante do *input* de cada utilizador. Desta forma, será possível otimizar e personalizar interações futuras de acordo com a experiência de cada cliente. Ainda relativamente aos resultados da fase de testes, implementar as sugestões dos voluntários como a capacidade de autocorreção, ter um menu mais completo e dar mais *feedback* ao utilizador.

Bibliografia

[1] Amazon Lex Overview. https://aws.amazon.com/lex/. Acedido em: 30/10/2018.

[2] Sobre a *Accenture*. https://www.*Accenture*.com/pt-pt/company. Acedido em: 01/11/2018.

[3] 2018 State of Chatbots Report. https://www.drift.com/wp-content/uploads/2018/01/2018-state-of-chatbots-report.pdf. Acedido em: 01/11/2018.

[4] Dialogflow Introduction - Overview. https://dialogflow.com/docs/intro. Acedido em: 01/11/2018.

[5] Dialogflow: *Google*'s chatbot platform doesn't disappoint. https://chatbottech.io/reviews/dialogflow.html. Acedido em: 02/11/2018.

[6] Machiraju S. Modi R. (2018) Conversations as Platforms. In: Developing Bots with Microsoft Bots Framework. Apress, Berkeley, CA

[7] Azure Bot Service. https://docs.microsoft.com/en-us/azure/bot-service/bot-service-overview-introduction?view=azure-bot-service-4.0. Acedido em: 02/04/2019.

[8] Amazon Lex features. https://aws.amazon.com/lex/features/. Acedido em: 03/11/2018.

[9] High, R., 2012. The era of cognitive systems: An inside look at IBM Watson and how it works. IBM Corporation, Redbooks

[10] Chatbot Development with IBM Watson. https://dzone.com/articles/chatbot-development-with-ibm-watson. Acedido em: 03/11/2018

[11] Saenz, J., Burgess, W., Gustitis, E., Mena, A. and Sasangohar, F., 2017. The Usability Analysis of Chatbot Technologies for Internal Personnel Communications. In IIE Annual Conference. Proceedings (pp. 1357-1362). Institute of Industrial and Systems Engineers (IISE).

[12] Dale, R., 2016. The return of the chatbots. Natural Language Engineering, 22(5), pp.811-817.

[13] About Pandorabots. https://www.pandorabots.com/docs/. Acedido em: 04/11/2018.

[14] Atwell, E., 2005. Web chatbots

[15] Repositório GitHub do ChatScript. https://github.com/ChatScript/ChatScript/. Acedido em: 06/11/2018.

[16] McNeal, M.L. and Newyear, D., 2013. Chatbot creation options. Library Technology Reports, 49(8), pp.11-17.

[17] Weizenbaum, J., 1966. ELIZA—a computer program for the study of natural language communication between man and machine. Communications of the ACM, 9(1), pp.36-45.

[18] Shawar, B.A. and Atwell, E., 2007. Chatbots: are they really useful?. In Ldv Forum (Vol. 22, No. 1, pp. 29-49).

[19] Wikipedia: ELIZA. https://en.wikipedia.org/wiki/ELIZA. Acedido em: 06/11/2018.

[20] AbuShawar, B. and Atwell, E., 2015. ALICE chatbot: trials and outputs. Computación y Sistemas, 19(4), pp.625-632.

[21] Shawar, B.A. and Atwell, E., 2002. A comparison between Alice and Elizabeth chatbot systems. University of Leeds, School of Computing research report 2002.19.

[22] Meet Mitsuki. https://www.pandorabots.com/mitsuku/. Acedido em: 06/11/2018.

[23] Aron, J., 2011. How innovative is Apple's new voice assistant, Siri?.

[24] Bellegarda, J.R., 2014. Spoken language understanding for natural interaction: The siri experience. In Natural Interaction with Robots, Knowbots and Smartphones (pp. 3-14). Springer, New York, NY.

[25] López, G., Quesada, L. and Guerrero, L.A., 2017, July. Alexa vs. Siri vs. Cortana vs. *Google* Assistant: a comparison of speech-based natural user interfaces. In International Conference on Applied Human Factors and Ergonomics (pp. 241-250). Springer, Cham.

[26] *Google* Assistant: Turning your house into a smart home with the *Google* Assistant. https://www.blog.*Google*/products/assistant/turning-your-house-smart-home-*Google*-assistant/. Acedido em: 07/11/2018.

[27] Alexa Voice Service Overview. https://developer.amazon.com/docs/alexa-voice-service/api-overview.html. Acedido em: 07/11/2018.

[28] Lei, X., Tu, G.H., Liu, A.X., Li, C.Y. and Xie, T., 2017. The insecurity of home digital voice assistants-amazon alexa as a case study. arXiv preprint arXiv:1712.03327.

[29] Cortana: Your intelligent assistant across your life. https://www.microsoft.com/en-us/cortana. Acedido em: 07/11/2018.

[30] Cortana explained: How to use Microsoft's virtual assistant for business. https://www.computerworld.com/article/3252218/collaboration/cortana-explained-why-microsofts-virtual-assistant-is-wired-for-business.html. Acedido em: 07/11/2018.

[31] Charland, A. and Leroux, B., 2011. Mobile application development: web vs. native. Queue, 9(4), p.20.

[32] What is the difference between a mobile app and a web app?. https://careerfoundry.com/en/blog/web-development/what-is-the-difference-between-a-mobile-app-and-a-web-app/. Acedido em: 12/11/2018.

[33] Native vs cross-platform app development. https://codeburst.io/native-vs-cross-platform-app-development-pros-and-cons-49f397bb38ac. Acedido em: 12/11/2018.

[34] Mobile app development: native vs web vs hybrid. https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/. Acedido em: 12/11/2018.

[35] SOAP and Service-Oriented Architecture. https://turbonomic.com/blog/on-technology/devs-venus-ops-mars-pt-3-soap-service-oriented-architecture/. Acedido em: 13/11/2018.

[36] Mumbaikar, S. and Padiya, P., 2013. Web services based on soap and rest principles. International Journal of Scientific and Research Publications, 3(5), pp.1-4.

[37] Devs are from venus, ops are from mars, pt. 3 - SOAP and Service-Oriented Architecture. https://turbonomic.com/blog/on-technology/devs-venus-ops-mars-pt-3-soap-service-oriented-architecture/. Acedido em: 13/11/2018.

[38] Garcia, C.M. and Abilio, R., 2017. Systems integration using web services, rest and soap: a practical report. Revista de Sistemas de Informação da FSMA, 1(19), pp.34-41.

[39] Leau, Y.B., Loo, W.K., Tham, W.Y. and Tan, S.F., 2012. Software development life cycle AGILE vs traditional approaches. In International Conference on Information and Network Technology (Vol. 37, No. 1, pp. 162-167).

[40] Agile VS Traditional development. http://www.professionalqa.com/agile-vs-traditional-development. Acedido em: 14/11/2018.

[41] Inc, B.R., 2003. Systems Development Lifecycle: Objectives and Requirements. New York: Queensbury.

[42] Dybå, T. and Dingsøyr, T., 2008. Empirical studies of agile software development: A systematic review. Information and software technology, 50(9-10), pp.833-859.

[43] Birgen, C., Preisig, H. and Morud, J., 2014. SQL vs. NoSQL. Norwegian University of Science and Technology, Scholar article.

[44] Parker, Z., Poe, S. and Vrbsky, S.V., 2013, April. Comparing nosql mongodb to an sql db. In Proceedings of the 51st ACM Southeast Conference (p. 5). ACM.

[45] Nass, C.I. and Brave, S., 2005. Wired for speech: How voice activates and advances the human-computer relationship (p. 9). Cambridge, MA: MIT press.

[46] Jensen, C., Farnham, S.D., Drucker, S.M. and Kollock, P., 2000, April. The effect of communication modality on cooperation in online environments. In Proceedings of the SIGCHI conference on Human Factors in Computing Systems(pp. 470-477). ACM.

[47] Skantze, G., 2007. Error Handling in Spoken Dialogue Systems-Managing Uncertainty, Grounding and Miscommunication. Gabriel Skantze.

[48] Gruhn, R.E., Minker, W. and Nakamura, S., 2011. Statistical pronunciation modeling for non-native speech processing. Springer Science & Business Media.

[49] Xamarin Mobile Development. Dickson, J., 2013. Xamarin Mobile Development. Acedido em 25/11/18.

[50] Microsoft Xamarin. https://visualstudio.microsoft.com/xamarin/. Acedido em 26/11/18.

[51] Gackenheimer, C., 2015. What is react?. In Introduction to React (pp. 1-20). Apress, Berkeley, CA.

[52] React Native: Bringing modern web techniques to mobile. https://code.fb.com/*Android*/react-native-bringing-modern-web-techniques-to-mobile/. Acedido em: 26/11/18.

[53] React Native on the Universal *Windows* Platform. https://blogs.*Windows*.com/buildingapps/2016/04/13/react-native-on-the-universal-*Windows*-platform/. Acedido em: 26/11/18.

[54] *Cordova* documentation. https://*Cordova*.apache.org/docs/en/latest/guide/overview/. Acedido em: 27/11/2018.

[55] Bosnic, S., Papp, I. and Novak, S., 2016, November. The development of hybrid mobile applications with Apache *Cordova*. In Telecommunications Forum (TELFOR), 2016 24th(pp. 1-4). IEEE.

[56] Turing, A.M., 2009. Computing machinery and intelligence. In Parsing the Turing Test (pp. 23-65). Springer, Dordrecht.

[57] The Loebner Prize. https://www.ocf.berkeley.edu/~arihuang/academic/research/loebner.html. Acedido em 29/11/2018.

[58] *Google*'s *Flutter* Mobile Development. https://www.smashingmagazine.com/2018/06/*Google*-*Flutter*-mobile-development/. Acedido em 29/11/2018.

[59] *Flutter* Docs: Introduction to widgets. https://*Flutter*.io/docs/development/ui/widgets-intro . Acedido em 29/11/2018

[60] Ali, M., Zolkipli, M.F., Zain, J.M. and Anwar, S., 2018, May. Mobile Cloud Computing with SOAP and REST Web Services. In Journal of Physics: Conference Series (Vol. 1018, No. 1, p. 012005). IOP Publishing.

[61] Hybrid apps vs Native apps: what should you build?. https://themanifest.com/app-development/hybrid-apps-vs-native-apps-which-should-you-build. Acedido em 12/11/2018.

[62] Shrawankar, U. and Thakare, V., 2010, March. Feature extraction for a speech recognition system in noisy environment: A study. In Computer Engineering and Applications (ICCEA), 2010 Second International Conference on (Vol. 1, pp. 358-361). IEEE.

[63] Rashad, M.Z., El-Bakry, H.M., Isma'il, I.R. and Mastorakis, N., 2010. An overview of text-to-speech synthesis techniques. Latest trends on communications and information technology, pp.84-89.

[64] Tokuda, K., Nankaku, Y., Toda, T., Zen, H., Yamagishi, J. and Oura, K., 2013. Speech synthesis based on hidden Markov models. Proceedings of the IEEE, 101(5), pp.1234-1252.

[65] Ze, H., Senior, A. and Schuster, M., 2013, May. Statistical parametric speech synthesis using deep neural networks. In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on (pp. 7962-7966). IEEE.

[66] The massive potential for chatbots in field service. https://www.ipctech.com/the-massive-potential-for-chatbots-in-field-service. Acedido em 11/12/2018.

[67] Can chatbots help the field service management industry?. https://www.fieldez.com/can-chatbots-help-field-service-management-industry/. Acedido em 12/12/2018.

[68] Aspect's Digital Self-service. https://www.aspect.com/solutions/self-service/digital-self-service/customer-service-chatbots. Acedido em 12/12/2018.

[69] StatCounter: Internet usage worldwide. http://gs.statcounter.com/press/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-worldwide. Acedido em 12/12/2018.

[70] Turing, A.M., 2009. Computing machinery and intelligence. In Parsing the Turing Test (pp. 23-65). Springer, Dordrecht

[71] Haapanen, P. and Helminen, A., 2002. Failure mode and effects analysis of software-based automation systems.

[72] Lebeuf, C., Storey, M.A. and Zagalsky, A., 2018. Software bots. IEEE Software, 35(1), pp.18-23.

[73] SearchCustomerExperience: Chatbot definition. https://searchcrm.techtarget.com/definition/chatbot. Acedido em 31/01/2019.

[74] Sansonnet, J.P., Leray, D. and Martin, J.C., 2006, August. Architecture of a framework for generic assisting conversational agents. In International Workshop on Intelligent Virtual Agents (pp. 145-156). Springer, Berlin, Heidelberg.

[75] npm: Node *MySQL* 2. https://www.npmjs.com/package/*MySQL*2#using-prepared-statements. Acedido em 29/04/2019.

[76] Ajuda NOS - Manuais e Equipamentos. https://www.nos.pt/particulares/ajuda/equipamentos-servicos/televisao/manuais-e-equipamento/Pages/default.aspx. Acedido em 03/05/2019.

[77] Web Speech API Specification. https://w3c.github.io/speech-api/. Acedido em 31/05/2019.

[78] Bot Framework Web Chat. https://github.com/microsoft/BotFramework-WebChat. Acedido em 31/05/2019.

[79] Redux Middleware. https://redux.js.org/advanced/middleware. Acedido em 31/05/2019.

[80] McTear, M.F., 2016, November. The Rise of the Conversational Interface: A New Kid on the Block?. In International Workshop on Future and Emerging Trends in Language Technology (pp. 38-49). Springer, Cham

[81] Chatbots in Customer Service - Statistics and Trends. https://www.invespcro.com/blog/chatbots-customer-service/. Acedido em 10/06/2019.

[82] Farber, D.A., Greer, R.E., Swart, A.D. and Balter, J.A., Cable and Wireless Internet Services Inc, 2003. Internet content delivery network. U.S. Patent 6,654,807.

[83] *MySQL* Documentation. https://dev.*MySQL*.com/doc/. Acedido em 10/05/2019.

[84] Chatbottest: Improve your chatbot's design. https://chatbottest.com. Acedido em 11/06/2019

[85] Rozga, S., 2018. Creating a New Channel Connector. In Pratical Bot Development (pp. 407-457). Apress, Berkeley, CA.

[86] Azure Bot Service. https://azure.microsoft.com/en-in/services/bot-service/. Acedido em 19/07/2019.

[87] Direct Line API 3.0 Reference. https://docs.microsoft.com/en-us/azure/bot-service/rest-api/bot-framework-rest-direct-line-3-0-api-reference?view=azure-bot-service-4.0. Acedido em 19/07/2019.

[88] Visual Studio Code. https://code.visualstudio.com/. Acedido em 19/07/2019.

[89] Sublime Text. https://www.sublimetext.com/.Acedido em 19/07/2019.

[90] Github. https://github.com/. Acedido em 19/07/2019.

[91] Bitbucket. https://bitbucket.org/product. Acedido em 19/07/2019.