Universidade de Aveiro
2019
Departamento de Eletrónica,
Telecomunicações e Informática

Tiago
Faria

ANÁLISE E REPRESENTAÇÃO DE FLUXOS DE EXECUÇÃO EM ASSISTENTES VIRTUAIS

Universidade de Aveiro
2019

Departamento de Eletrónica,
Telecomunicações e Informática

Tiago Faria

ANÁLISE E REPRESENTAÇÃO DE FLUXOS DE EXECUÇÃO EM ASSISTENTES VIRTUAIS

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor Hélder Troca Zagalo, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Joaquim Arnaldo Carvalho Martins
professor catedrático da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Fernando Joaquim Lopes Moreira
professor catedrático do Departamento de Ciência e Tecnologia da Universidade Portucalense

Prof. Doutor Hélder Troca Zagalo
professor auxiliar da Universidade de Aveiro

agradecimentos /
acknowledgements

Quero agradecer aos meus pais, família e namorada por me terem acompanhado neste percurso desde o primeiro instante. Acreditaram sempre em mim e no meu potencial, incentivaram-me, deram-me força e sobretudo conseguiram tranquilizar-me em vários pontos-chave desta fase decisiva da minha vida. É graças a vós e ao vosso apoio que consegui construir a pessoa que sou hoje, capaz de alcançar várias metas e de agarrar outras oportunidades.

A todos os meus amigos e colegas, não só por quererem acompanhar de perto o meu percurso académico, como também por ficarem entusiasmados pelo mesmo. Sinto-me congratulado por conseguir ser um exemplo para alguns e por também ter tido outras pessoas como referência.

Ao Professor Hélder Zagalo, orientador desta dissertação, por ter acreditado em mim e nas minhas capacidades, aquando o lançamento da proposta. Agradeço também por toda a disponibilidade, apoio prestado, assim como interesse e preocupação no meu trabalho.

Ao meu orientador na Altice Labs, o Engenheiro Jorge Monteiro, por me ter dado esta oportunidade de integrar um projeto inovador e ambicioso na empresa. A ele, o meu obrigado por confiar nas minhas aptidões, lançando-me vários desafios que fizeram com que mostrasse a minha capacidade de trabalho, motivando-me para ser e fazer melhor, dia após dia.

Aos meus colegas de equipa, por me receberem de braços abertos e encontrarem-se disponíveis para qualquer problema ou dúvida da minha parte, fosse algo de índice profissional ou pessoal.

A todos, o meu profundo e sincero obrigado.

Palavras Chave

assistentes virtuais, análise de texto, reconhecimento de entidades, processamento de linguagem natural, compreensão de linguagem natural, inteligência artificial, plataformas *web*.

Resumo

A tecnologia tem avançado colossalmente, assim como a sua aquisição. Existe atualmente uma vasta gama de oferta no mercado com diferentes tarefas e propósitos, onde os acessórios eletrónicos e os assistentes virtuais têm aumentado o seu número de vendas. Estes realçam-se pelo facto de terem um comportamento aproximado ao do humano. Desta forma, a análise e o estudo destes mesmos assistentes, assim como a aposta na interpretação semântica de texto, apresenta um trabalho e uma oferta mais complexa e de melhor qualidade, resultando numa melhor experiência para os utilizadores. Esta dissertação tem como principal objetivo o estudo, a análise e a implementação das melhores metodologias para o desenvolvimento de um assistente virtual, recorrendo a áreas como a Inteligência Artificial, Análise de Texto e Processamento de Linguagem Natural, *Speech-to-Text* e por fim, Plataformas *Web*, de forma a apresentar as sugestões dadas para complementar o assistente.

Keywords

virtual assistants, text analysis, named entity recognition, natural language processing, natural language understanding, artificial intelligence, web platforms.

Abstract

Technology has advanced colosally as well as its acquisition. Currently, there is a wide range of offers in the market with different tasks and purposes, where gadgets and virtual assistants have increased their number of sales. These are highlighted by the fact that they have an approximate behavior of the human. In this way, the analysis and the study of these same assistants, as well as the focus on semantic interpretation of text, presents a more complexe work and a higher quality, resulting in a better user experience. This dissertation has as main objective the study, analysis and implementation of the best methodologies for devoloping a virtual assistant, using areas such as Artificial Intelligence, Text Analysis and *Natural Language Processing*, and finally Web Plataforms, in order to present the suggestions to complement the assistant.

Conteúdo

Lista de Figuras

Lista de Tabelas

Glossário

API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Networks
CRF	Conditional Random Field
CRUD	Create, Read, Update and Delete
DL	Deep Learning
DER	Diarization Error Rate
DOM	Domain Object Model
FAQ	Frequently Asked Questions
GATE	General Architecture for Text Engineering
HATEOAS	Hypermedia As The Engine Of Application State
HMM	Hidden Markov Models
IA	Inteligência Artificial
IoT	Internet Of Things
JSX	JavaScript XML
KNN	K-Nearest Neighbors
LPC	Linear Predictive Coding
LUIS	Language Understanding
ML	Machine Learning
MFCC	Mel Frequency Cepstral Coefficient
MLM	Masked Linguistic Model
MVC	Model View Controller
MVVM	Model View View-Model
NER	Named Entity Recognition
NLG	Natural Language Generation
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NLU	Natural Language Understanding
NPM	Node Package Manager
NSP	Next Step Prediction
PLP	Perceptual Linear Predictive
POS	Part-of-Speech
REST	Representational State Transfer
RDF	Resource Description Framework
RNN	Recurrent Neural Networks

SD	Speaker Diarization
SPA	Single Page Application
STT	Speech-to-Text
SVM	Support Vector Machine
tf-idf	Term Frequency - Inverse Document Frequency
UIS-RNN	Unbounded Interleaved-State Recurrent Neural Networks
V-DOM	Virtual DOM
WebCT	Web Component Toolkit

CAPÍTULO 1

Introdução

1.1 Enquadramento

Nos dias de hoje, as soluções desenvolvidas têm apresentado um grau de complexidade cada vez mais elevado, fazendo com a tecnologia se encontre num ciclo de constantes mudanças e consequentes melhorias. Com esta evolução, o grau de exigência é cada vez maior e os padrões de qualidade são cada vez mais restritos. Mediante as necessidades atuais da sociedade e a forte investigação e respetivo investimento em soluções cada vez mais inovadoras e interativas, existem várias metodologias, convenções e avaliações que se preocupam em oferecer e sobretudo assegurar um produto e/ou serviço que apresente métricas de qualidade satisfatórias e que também garanta a correta execução do mesmo, de modo a que não só solucione um problema existente, como também garanta uma boa experiência para os seus utilizadores.

No meio de dispositivos populares como *smartphones*, *smartwatches*, *tablets, smart TVs* e outro tipo de tecnologias, e de áreas como a Realidade Virtual e Aumentada, o comércio eletrónico, a *Internet Of Things (IoT)* e a Inteligência Artificial, encontram-se os assistentes virtuais. Um assistente virtual inteligente é uma aplicação que tem a capacidade de processar linguagem natural humana, criando conversas com o seu utilizador, de forma a cumprir os pedidos feitos pelo mesmo. Estes dispositivos suportam uma vasta quantidade de tarefas, desde a gestão de *e-mails*, agendamento de compromissos e alterações de diversas configurações, como a possibilidade de controlar uma casa autónoma e inteligente, administrando recursos como a água, a eletricidade e o gás, entre outras funcionalidades possíveis. Apesar de conseguir enquadrar-se em diversos casos de uso, os assistentes virtuais são mais conhecidos para o uso pessoal, inseridos num ambiente doméstico. Exemplos disso são o Google Assistant [1], a Siri [2], a Amazon Alexa [3] e o Microsoft Cortana [4].

Um dos principais problemas nos assistentes virtuais e no processamento de linguagem natural humana refere-se ao facto de haver uma tendência de maior foco na análise sintática, descurando a parte semântica. Como consequência disso, muitos dos sistemas existentes não

conseguem processar mensagens com conteúdo semelhante, porém, com uma estrutura sintática diferente. Assim, torna-se importante analisar o texto, tendo em conta a contextualização do mesmo, tal como a classificação de entidades e categorias existentes.

Para dar uma resposta fundamentada ao desafio proposto pela Altice Labs, pretende-se recorrer a algoritmos e técnicas referentes ao processamento de linguagem natural e ao reconhecimento de entidades. É também requerida a utilização da plataforma *web* desenvolvida pela própria empresa, a WebCT, de forma a gerar e a analisar o comportamento inteligente do assistente e, sugerir posteriormente, o enriquecimento do mesmo.

1.2 Objetivos Principais

O objetivo desta dissertação passa por analisar, definir e implementar os melhores métodos para fazer a análise semântica do texto e o respetivo reconhecimento e classificação de entidades. Para isso, serão implementadas várias técnicas de processamento de linguagem natural, que após a criação de um fluxo de execução, focado num determinado domínio de problema, serão utilizadas para avaliar os diálogos que não foram corretamente interpretados pelo assistente. Desta forma, é requisito a criação de uma página *web*, onde seja possível observar os resultados da análise de texto feita aos diálogos anteriores, podendo assim escolher o tipo de ação a realizar, mediante o resultado obtido.

Por forma a atingir estes objetivos, foi também definido o desenvolvimento de um bloco, que para além de enquadrar-se no contexto do problema, pode ser utilizado de forma independente. Mediante um conjunto de registos de áudio, com a participação de um operador e um cliente, estipulou-se a transcrição automática de cada conversa, separada e associada aos seus dois intervenientes. Para tal, definiu-se a utilização de ferramentas como o *Speech-to-Text* e a *Speaker Diarization*.

1.3 Metodologia de Trabalho

Para a realização desta dissertação, foi adotada uma metodologia de trabalho que englobou as seguintes fases:

1. **Revisão de Literatura**
   - discussão e compreensão das áreas do domínio do problema;
   - revisão de literatura de cada tema abrangido pelo projeto;
   - estudo e análise de *frameworks* já existentes;
   - realização de tutoriais, experiências e testes às ferramentas e metodologias consideradas;
   - balanço das vantagens e desvantagens das tecnologias, mediante o tipo de projeto e aplicação;
   - elaboração da teoria de conceito.
2. **Desenvolvimento das transcrições de áudio**

- recolha e análise das amostras fornecidas;
- transcrição manual dos ficheiros de áudio;
- integração das biblioteca de aplicação da *Speaker Diarization*;
- divisão dos ficheiros de áudio, mediante os intervalos de tempo associados a cada orador;
- implementação do reconhecimento da fala;
- comparação dos resultados obtidos com as transcrições manuais.

3. **Construção do identificador de entidades**
   - criação do *corpus* de dados;
   - implementação de metodologias referentes à identificação de entidades;
   - melhorias na *performance* do identificador.
4. **Criação de micro-serviços**
   - criação de uma serviço para as transcrições de áudio;
   - criação de um servidor temporário para o *download* das transcrições;
   - criação de um serviço para o identificador de entidades;
   - criação de um serviço centralizado para a gestão dos serviços referentes às transcrições e entidades, assim como outras funcionalidades;
   - implementação do serviço de sugestões de intenções.
5. **Criação de páginas *web***
   - criação de um componente na WebCT, de forma a ter uma interface do serviço anteriormente criado.
6. **Escrita da dissertação, contemplando o trabalho desenvolvido**

1.4 Contribuição

A principal contribuição desta dissertação consistiu em criar os seguintes subsistemas:

- um dos sistemas é encarregue de utilizar as metodologias de *Speech-to-Text* e *Speaker Diarization*, transformando as gravações de áudio em texto. Com esta aplicação, é possível compreender o que é dito pelo utilizador, como é também possível interpretar ficheiros de áudio, de forma a substituir o processo manual das transcrições, poupando recursos e tempo;
- o outro subsistema é encarregue de fazer a análise de texto, através do processamento da linguagem natural humana. Por norma, um assistente possui um fluxo de execução, com foco num determinado domínio de problema. Com a criação dos dados necessários e após o treino dos mesmos, é possível manter uma conversa inteligível, contudo, haverão questões feitas pelo utilizador que não serão compreendidas por parte do assistente. Quando tal situação acontece, é acionado um novo módulo que trata de manipular o texto e reconhecer entidades, onde são feitas sugestões de associações entre a frase incompreendida e as categorias reconhecidas. Tal funcionalidade permite aumentar o conhecimento do assistente, e consequentemente, diminuir o número de diálogos mal interpretados.

1.5 Organização da Dissertação

A presente dissertação encontra-se dividida em sete capítulos, sendo o atual e primeiro capítulo dedicado ao enquadramento, objetivos principais, metodologias seguidas e contribuição dada à solução do desafio proposto.

O segundo, terceiro e quarto capítulo enquadram-se no estado da arte da dissertação, descrevendo a revisão de literatura e investigação feita, de forma a fortalecer o conceito criado. No segundo capítulo são apresentados conceitos sobre o *Speech-to-Text* e algumas das suas técnicas de aplicação, enquanto no terceiro são expostos casos de estudo e ferramentas que permitam a análise de texto. No quarto e último capítulo desta secção, são analisadas as tecnologias para a construção de interfaces *web*.

O quinto capítulo descreve todos os detalhes relativos à solução proposta, apresentando a sua arquitetura e os passos dados na sua implementação, enquanto o sexto capítulo apresenta os resultados obtidos e experiências efetuadas.

No último e sétimo capítulo são enunciadas algumas conclusões, juntamente com as lacunas encontradas e o trabalho proposto para o futuro. No final, é apresentada a bibliografia consultada, onde estão agrupadas todas as referências para a elaboração desta dissertação, desde artigos, documentos, livros e páginas *web*.

CAPÍTULO 2

Speech-to-Text

Neste capítulo são apresentados métodos que permitam fazer transcrições de registos de diálogos gravados entre operadores e clientes, de forma a criar um conjunto de dados enriquecido, que servirá de entrada para os classificadores de entidades e respetivas ações. Para a análise de texto conseguir alcançar resultados satisfatórios, é necessária a criação de um *corpus* de dados consistente, face a um domínio de problema específico. Para tal, e como o domínio do problema tem bastante foco na área de apoio ao cliente, ou seja, na habilidade dos utilizadores interagirem através de um diálogo, surge a necessidade de recolher informação dos padrões linguísticos dos mesmos.

2.1 Conceitos sobre *Speech-to-Text*

Segundo os dicionários da Oxford, a comunicação define-se como a transmissão e a troca de informação perante a utilização de um meio de comunicação, entre duas ou mais pessoas [5]. Estes meios dividem-se na comunicação escrita através de mensagens, livros e jornais, na comunicação oral através da fala, telefone e rádio, na comunicação áudio-visual como a televisão e o cinema e na comunicação física através da linguagem gestual.

Dos meios de comunicação anteriormente descritos, a fala é a mais utilizada, e consequentemente, a mais importante entre os humanos. Desta forma, surge a motivação e a necessidade das máquinas dialogarem com os humanos através da linguagem natural. Para este desafio e de modo a corresponder a um comportamento inteligente, é necessário haver a perceção natural do ambiente inserido, tal como a habilidade de perceber determinadas ações e estímulos, à semelhança do que acontece no comportamento humano [6].

O *Speech-to-Text (STT)* é o processo de reconhecimento de um discurso durante o áudio analisado, construindo uma transcrição como resultado final. Por outras palavras, estes sistemas recebem como parâmetro a fala, emitida por um humano, transformando-a num

conjunto de palavras associadas, onde o seu objetivo passa por extrair, caracterizar e reconhecer informações relevantes sobre a mesma [7].

Mediante a voz emitida e os dados de entrada, os sistema de reconhecimento da fala podem ser classificados da seguinte forma [8]:

- **Sistemas dependentes e independentes do orador**: nos sistemas dependentes, o discurso de um utilizador é usado para treino, de modo a reconhecer posteriormente a sua fala. Por outro lado, os sistemas independentes podem reconhecer qualquer voz, pois estes não são sensíveis a nenhum utilizador em particular;
- **Reconhecimento de fala isolado e contínuo**: num sistema de reconhecimento de fala isolado, o orador fala pausadamente, com intervalos precisos entre duas palavras, já no sistema contínuo, não há a definição prévia de qualquer tipo de espaço temporal. No nosso quotidiano, as conversas têm por hábito serem contínuas, fazendo com que um sistema de reconhecimento contínuo seja o mais adequado ao domínio do problema. No entanto, o reconhecimento feito de forma isolada é mais simples, devido à indicação clara dos pontos de começo e fim dos intervalos;
- **Baseados em palavras e sub-palavras**: os sistemas de reconhecimento podem ser treinados para identificar palavras completas, sendo estes dependentes de palavras-chave, ou então podem reconhecer sílabas e fonemas, criando desta forma um sistema dependente de sub-palavras.

Em relação ao seu *design*, o reconhecimento de áudio lida com as seguintes decisões e problemas [8]:

- **Seleção e avaliação das características**: no reconhecimento da fala, existe um conjunto de características que permitem distinguir as várias unidades linguísticas existentes. Estas podem ser articulatórias (ações dos órgãos da voz), acústicas (amplitude, frequência e potências) ou percetivas (perceção do som pelos ouvidos e cérebro). Geralmente, a sua seleção é feita no pré-processamento de grandes quantidades de dados, ou seja, é analisada a forma de escolher um subconjunto, necessário para construir modelos, de forma a capturar as características chave na voz, onde são utilizadas técnicas como *Linear Predictive Coding (LPC)*, que faz a aproximação entre a amostra de som atual e as amostras anteriores, através de combinações lineares, *Mel Frequency Cepstral Coefficient (MFCC)* para representar, a curto prazo, o espectro auditivo de um sinal de fala, através do cálculo da transformação linear do cosseno e *Perceptual Linear Predictive (PLP)* para a modelação do espectro anterior [9];
- **Escolha da regra de decisão**: os padrões dos sinais de áudio são tratados como características da voz, onde estas são utilizadas para tomar decisões na seleção das palavras. Por norma, os vetores de características estão associados a um conjunto de palavras em particular, contudo, existem alguns casos onde os mesmos não conseguem identificar nenhuma palavra. Quando esta situação acontece, a mesma é vista como

um erro de reconhecimento, explicando assim a necessidade de haver probabilidades baseadas nas regras de decisão e critérios de avaliação das características.

O STT é um meio de comunicação intuitivo, que terá a longo prazo grandes efeitos na facilidade de interação entre os humanos e as máquinas [6]. Para além da economia de tempo e esforço humano no processo das transcrições de áudio para texto, este processo é maioritariamente utilizado na automação de casas e em propósitos médicos, nomeadamente na simplificação de algumas tarefas básicas para pessoas idosas ou com alguma incapacidade física e/ou visual [8]. Contudo, a transformação de voz em texto encontra-se ainda em constante estudo e melhoria, visto que continua a ser algo não completamente trivial, dada a frequente dificuldade em captar som quando as condições externas se deterioram. Com isto, e mediante a importância da sua implementação, foram feitos investimentos na investigação e desenvolvimento de sistemas que processam automaticamente a fala, como por exemplo, sistemas de melhoria de voz, síntese e compreensão da fala, reconhecimento de oradores através do *Audio Fingerprinting*, capaz de identificar uma amostra de áudio, semelhante a outra já treinada e reconhecimento e verificação da fala [7].

2.2 *Speaker Diarization*

Em grande parte dos cenários do mundo real, os discursos realizados não vêm com os segmentos de áudio bem definidos, onde existe apenas um único orador (*speaker*). Para além disso, é bastante frequente a existência de interrupções numa conversa, originadas pelos seus participantes, fazendo com que a divisão do áudio perante a fala não seja trivial [10]. Com a necessidade de identificar várias pessoas durante uma conversa, de forma a criar uma ligação entre as frases mencionadas, é assim requerida a aplicação do paradigma da *Speaker Diarization (SD)* (Diarização do Orador).

A *SD* é a tarefa de dividir uma *stream* de áudio ou vídeo em segmentos temporais homogéneos, de acordo com a identidade do orador [11]. Por outras palavras, esta metodologia permite inferir quem fala e quando, numa gravação, identificando para cada pessoa existente, as regiões onde a mesma discursa [12].

A identificação de cada pessoa é feita de uma forma não-supervisionada, ou seja, são geradas associações entre as regiões temporais e vários rótulos, podendo estes representar a identidade de uma pessoa, ou o seu género, o tipo de canal de comunicação utilizado (largura de banda larga ou de banda estreita), o tipo de ambiente de fundo (barulhento, sossegado, ruído, música, etc.), assim como outras características presentes no sinal. Mais especificamente, a SD trata de identificar o número de oradores presentes, sem qualquer conhecimento à priori dos mesmos [13].

Quanto à arquitetura da SD, a mesma é constituída pelos seguintes módulos base [13]:

- **Extração de características**: aquisição dos dados captados através do sinal de voz emitido e extração de características acústicas importantes como a MFCC e a PLP;
- **Deteção da fala**: deteção dos eventos da fala e da não-fala;
- **Segmentação do Orador**: componente que ajuda a dividir o sinal da fala em segmentos semelhantes de um determinado orador, detetando também a mudança do mesmo para outra pessoa;
- ***Clustering* do Orador**: algoritmos de identificação e agregação dos dados do mesmo orador, tornando-os num único *cluster*.

Nestes sistemas, existem duas abordagens possíveis: *Bottom-Up* e *Top-Down*. A *Bottom-Up*, também conhecida como *clustering* aglomerado hierárquico, é uma abordagem que treina inicialmente um dado conjunto de *clusters* ou modelos, ajudando na fusão e sucessiva redução do número de *clusters*, até restar apenas um para cada orador. Sendo a abordagem mais comum, a mesma consiste nos seguintes passos:

1. Inicialização dos *clusters* com os segmentos de fala;
2. Cálculo das distâncias entre os pares de *clusters*;
3. Fusão dos *clusters* mais próximos;
4. Atualização das distâncias dos restantes *clusters*;
5. Repetição dos últimos três passos, até que seja encontrado um critério de paragem.

Quanto à *Top-Down*, esta é uma abordagem muito menos popular que faz, como primeiro passo, a modelação da *stream* de áudio completa com apenas um único modelo de um orador, adicionando sucessivamente novos modelos, até que o número total de pessoas existentes seja contabilizado [13].

Dado o tipo de processamento e identificação feita pela SD, a mesma pode deparar-se com alguns desafios que dificultam a classificação correta dos seus oradores. Através de alguns estudos feitos na área, comprovou-se que a sobreposição da fala é uma das fontes mais problemáticas, ou seja, fontes de áudio provenientes de reuniões com um vasto leque de participantes contém uma maior frequência de interrupções, sobreposições entre os oradores, assim como o ruído envolvido no ambiente em causa, tendo como consequência o aumento da probabilidade de erro [14]. A acrescentar a estas dificuldades obtidas durante este processamento, surgem outras situações como a mudança de voz por parte dos oradores, derivada do tipo de ambiente em que se encontram inseridos, a variação do estado do espírito dos mesmos e o modo de comunicação feito, assim como o tamanho da largura de banda (no caso dos telefones), as várias configurações de gravação existentes, a curta duração dos segmentos de fala e a ocorrência de vários eventos não-vocais [15][13].

2.3 Tecnologias Existentes

De forma a conseguir identificar os diferentes oradores presentes numa conversa, assim como os segmentos de fala de cada um, torna-se necessária a utilização de uma biblioteca que consiga

aplicar a metodologia da SD. Assim, são apresentadas algumas tecnologias capazes de fazer o procedimento e análise de áudio.

2.3.1 ALIZE

A ALIZE é uma biblioteca colaborativa e de acesso livre, desenvolvida em C++ e dedicada ao reconhecimento da fala, em que o seu principal objetivo consiste no fornecimento de um conjunto de ferramentas de alto e baixo nível, de forma a gerir várias tarefas na área do reconhecimento de oradores.

Seguindo a metodologia orientada a objetos e com o intuito de ir ao acordo das necessidades de todos os utilizadores, esta biblioteca foi desenvolvida numa arquitetura multi-camada. A camada base é a *ALIZE-Core*, que é uma biblioteca de baixo nível, onde estão agrupadas todas as funções necessárias para as misturas Gaussianas, assim como funções de I/O para os vários formatos de ficheiros. Em cima desta mesma camada encontra-se uma ferramenta que oferece funcionalidades de alto nível, a *LIA-RAL*, que é composta pelos seguintes componentes [16]:

- ***LIA-SpkDet***: conjunto de ferramentas requerido pelos sistemas de autenticação do orador;
- ***LIA-SpkSeg***: ferramentas para a aplicação da SD;
- ***LIA-Utils***: utilidades para manipular os vários formatos de áudio utilizados;
- ***LIA-SpkTools***: biblioteca que serve de base para outras partes, fornecendo um conjunto de funções de alto nível no topo da *ALIZE-Core*.

A *LIA-RAL* é também composta pelo *SimpleSpkDetSystem*. Este módulo, executado paralelamente aos outros módulos da mesma camada, oferece uma *Application Programming Interface (API)* simples e de alto nível para programadores que pretendam desenvolver a identificação e verificação de oradores, inserido nas suas aplicações. A próxima figura ilustra a arquitetura multi-camada da ALIZE.

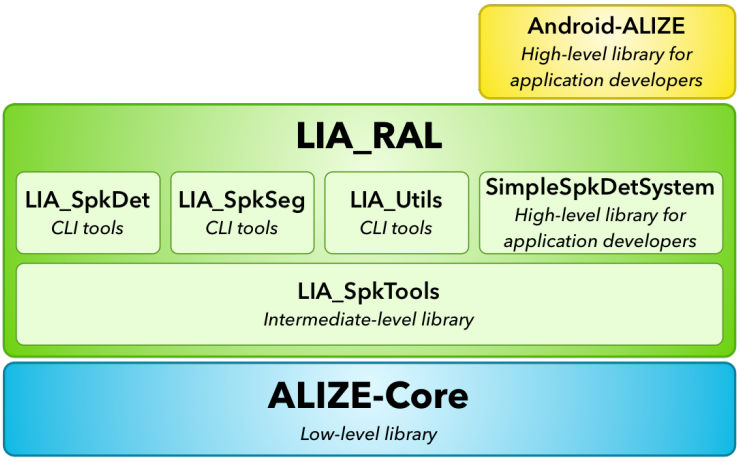

Figura 2.1: Arquitetura da ALIZE [16]

Sendo uma arquitetura constituída por várias camadas, a biblioteca em causa disponibiliza as funcionalidades através de vários servidores, nomeadamente o servidor de características, que gere os dados acústicos, o servidor das misturas, que lida com os modelos de armazenamento, modificação, ligação entre os componentes e escrita e/ou leitura e o servidor encarregue de implementar estimadores estatísticos. Com esta abordagem, são alcançadas as seguintes vantagens [17]:

- Cada servidor é acessível, graças a uma pequena lista de funcionalidades de alto nível, enquanto as de baixo nível são abstraídas para o utilizador;
- Devido à independências dos servidores, todos são otimizados e atualizados separadamente;
- Várias instâncias do mesmo servidor podem ser lançadas ao mesmo tempo;
- O código desenvolvido pelo utilizador apresenta a mesma estrutura, organizada entre os servidores principais, ajudando assim na compreensão e desenvolvimento do código fonte;
- A arquitetura implementada permite que várias instância do mesmo servidor possam ser lançadas em paralelo em terminais distintos, de modo a aumentar o poder computacional.

2.3.2 UIS-RNN

A *Unbounded Interleaved-State Recurrent Neural Networks (UIS-RNN)* é uma biblioteca desenvolvida com o intuito de resolver os problemas de segmentação e *clustering* de dados sequenciais, onde a aprendizagem é feita através da incorporação dos exemplos fornecidos [18]. Sendo algo que é realizado de forma completamente supervisionada, esta biblioteca realiza tarefas do reconhecimento de fala com uma abordagem relativamente diferente às tecnologias concorrentes na mesma área.

Os sistemas atuais que aplicam a SD são tipicamente baseados em algoritmos de *clustering* como o K-Means ou o *Clustering* Espectral. Visto que estes métodos não são supervisionados, os mesmos não conseguem usufruir dos dados anotados, referentes aos oradores existentes. Para além disso, este tipo de algoritmos, executados *online* e em tempo real, costumam ter pior desempenho e eficácia na separação dos discursos durante o processamento das *streams* de áudio de entrada. A principal diferença entre a UIS-RNN e estes algoritmos, é o facto de todas as características dos oradores serem modeladas por um parâmetro partilhado nas *Recurrent Neural Networks (RNN)*, que são uma classe das redes neuronais artificiais, onde as conexões entre os nós formam um grafo orientado ao longo de uma sequência temporal. Assim, todos os oradores são distinguidos por utilizarem diferentes estados das RNN, intervalados entre o domínio de tempo [19]. Por outras palavras, cada orador inicia o processo com uma instância própria, sendo o estado inicial partilhado por todos e mais tarde atualizado especificamente para cada um, mediante as características presentes no áudio. Ou seja, cada instância atualiza o seu estado continuamente, até que seja identificado um outro orador, havendo a possibilidade do mesmo estado ser posteriormente atualizado, desde que o orador anterior volte a falar novamente.

Representar os oradores através das RNN permite obter um conhecimento de alto nível, partilhado entre estes e as suas falas associadas, através da utilização dos parâmetros da rede neuronal. Desta forma, a UIS-RNN oferece uma metodologia completamente supervisionada, onde é possível estimar o número de oradores presente, assim como um melhor aproveitamento dos dados anotados criados [19][18].

2.3.3 Pydub

A Pydub é uma biblioteca *open-source*, escrita em Python, que permite fazer manipulação de áudio. De fácil acesso e compreensão, esta biblioteca fornece uma interface de alto nível, sendo possível realizar diversas tarefas onde certos pormenores são abstraídos ao utilizador [20]. Mediante a leitura de um ficheiro, ou um intervalo de tempo de uma gravação, os mesmos são interpretados como sendo um segmento de áudio, possibilitando a aplicação das seguintes funcionalidades:

- Leitura e reprodução de áudio;
- Processamento de sinal com compressão, equalizador, normalização e modificador de velocidade;
- Geradores de sinais de áudio;
- Sistema de registos de efeitos;
- Divisão de ficheiros e/ou segmentos, mediante um intervalo de tempo;
- Remoção de silêncio e divisão de áudio através do mesmo.

A Pydub é também flexível no suporte aos diferentes formatos dos ficheiros de áudio, permitindo a importação, exportação e conversão entre eles.

2.3.4 pyAudioAnalysis

A pyAudioAnalysis é uma biblioteca *open-source*, escrita em Python, que é dedicada ao processamento da análise do áudio. O objetivo desta tecnologia consiste no fornecimento de um conjunto de funcionalidades, através de um *design* compreensível e fácil de se usar, que podem ser aplicadas em diversos domínios como a automação de casas inteligentes, através da deteção de áudio, o reconhecimento da emoção da fala, a classificação de uma possível depressão humana, baseada nas características auditivas, a segmentação de gravações musicais, a recomendação de filmes, com base em conteúdos multimodais, assim como a monitorização dos hábitos alimentares de um ser humano.

A pyAudioAnalysis é composta pelas seguintes funcionalidades principais: extração de características, treino e aplicação de classificadores de áudio, regressão, segmentação e visualização. A Figura 2.2 demonstra um diagrama geral da composição desta biblioteca.

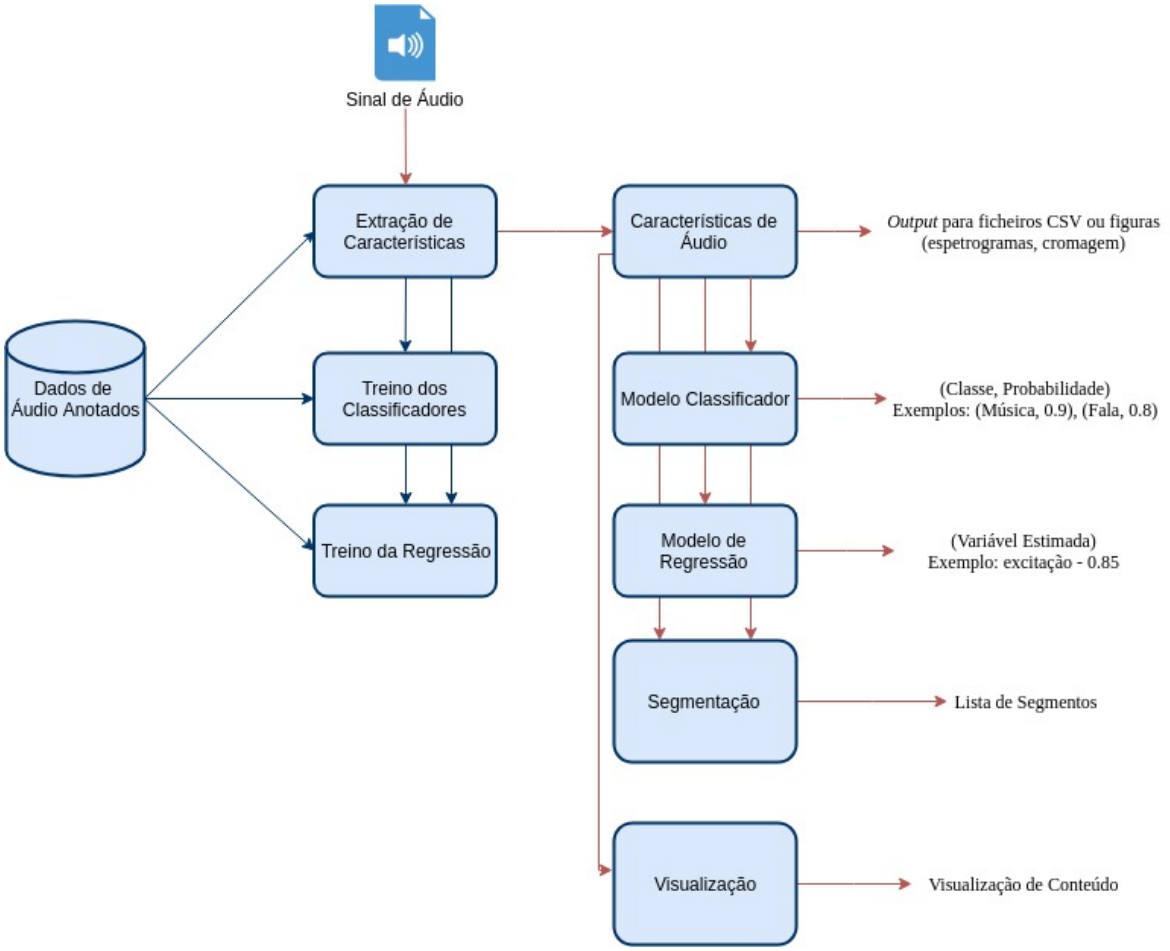

Figura 2.2: Diagrama Geral da pyAudioAnalysis [21]

Na extração de características, é feita a recolha de vários dados provenientes do áudio, ambos do domínio do tempo e da frequência, onde as características do tempo são diretamente recolhidas através da amostra do sinal, enquanto as da frequência são baseadas na magnitude da Transformada Direta de Fourier. Em termos temporais, as mesmas características também podem ser extraídas, com base no curto prazo de tempo, onde o sinal de áudio é primeiramente dividido em *frames* (20 a 100 milissegundos), onde para cada uma são calculadas todas as 34 características de áudio disponíveis na biblioteca, como a MFCC, a entropia da energia, os vetores croma, a entropia espectral, entre outros. Por outro lado, na análise de áudio de média ou longa duração, o sinal é dividido em segmentos de médio prazo (1 a 10 segundos). No caso das gravações de longa duração, como uma música, pode ser aplicada uma média dos cálculos obtidos a médio prazo, tal que todo o sinal possa ser representado por um vetor que consiste na média das características a médio prazo [21].

A classificação é utilizada na atribuição de uma amostra desconhecida a um conjunto de classes pré-definidas, através do treino de modelos supervisionados, que classificam, tanto os segmentos como os conjuntos de áudio inteiros. Para este fim, a pyAudioAnalysis adota os modelos *Support Vector Machine (SVM)* e *K-Nearest Neighbors (KNN)*, onde é também

implementada uma validação cruzada, de forma a extrair o classificador com os parâmetros otimizados, como por exemplo, o custo do parâmetro no SVM ou o número de vizinhos mais próximos usados no KNN. O processo da extração é também embutido na classificação, tal que os utilizadores possam classificar diretamente os ficheiros de áudio desconhecidos ou até ficheiros armazenados em alguma pasta em particular [21].

Na regressão, é estimado o valor de uma variável desconhecida, dado um vetor de características associado. Esta funcionalidade pode ser também importante no contexto da aplicação da análise de áudio, onde é feito o mapeamento desde as características de áudio até às variáveis de valor real. Um exemplo típico é a estimativa da emoção do discurso, onde as emoções não são representadas por classes discretas. A biblioteca em causa suporta o treino de regressão em SVM, em ordem do mapeamento das características de áudio para uma ou várias variáveis supervisionadas [21].

A segmentação foca-se em dividir um sinal de áudio contínuo em segmentos de conteúdo homogéneo. Neste contexto, são fornecidas duas soluções na segmentação do áudio: algoritmos supervisionados e algoritmos não-supervisionados ou semi-supervisionados. Nos algoritmos supervisionados, é adotado um conhecimento à priori, havendo um esquema de classificação pré-treinada. Para este tipo de algoritmos, é oferecida uma segmentação conjunta de tamanho fixo através de métodos baseados em *Hidden Markov Models (HMM)* [22]. Por outro lado, na segmentação não-supervisionada ou semi-supervisionada, não existe qualquer tipo de conhecimento antecipado sobre as classes envolvidas do conteúdo do áudio que é usado. Exemplos de aplicação destes algoritmos são a remoção de silêncio, a SD e o *audio thumbnailing*, que consiste na extração da amostra mais representativa de uma gravação musical.

A visualização é utilizada para extrair e observar as relações de conteúdo entre uma coleção de registos de áudio, sendo assim possível retirar algumas conclusões úteis através de gráficos, esquemas e conteúdos visuais interativos.

Além das funcionalidades principais, a pyAudioAnalysis oferece as seguintes características que, combinadas como um todo, são únicas, em comparação com outras bibliotecas:

- Ligação entre a extração de características e os componentes conceptuais de *Machine Learning*, de forma a complementar soluções de classificação e segmentação de áudio;
- Implementação das técnicas mais modernas, assim como técnicas base, de forma a resolver tarefas de áudio amplamente utilizadas;
- Fornecimento de modelos pré-treinados para algumas tarefas supervisionadas como a distinção entre a fala e a música, a classificação do género musical e a deteção de eventos cinematográficos;
- Todas as funcionalidades fornecidas são escritas utilizando código simples e distinto, tal que os passos dos algoritmos conceptuais possam ser claramente apresentados.

2.3.5 Comparativo

Na Tabela 2.1 é feita uma síntese das propriedades das tecnologias anteriormente apresentadas.

Linguagem

A maioria das tecnologias analisadas utiliza o Python, com exceção da ALIZE, que usufrui do C++.

Funcionalidades Principais

A ALIZE, a UIS-RNN e a pyAudioAnalysis têm capacidade para fazer o processamento, análise e segmentação do áudio com a SD, assim como o reconhecimento da fala através do STT. Visto que a Pydub é focada apenas na manipulação do som, esta não suporta tais funcionalidades, fornecendo operações como a aplicação de efeitos e a modificação do volume e da velocidade.

Tipo de Classificação

Quase todas as tecnologias apresentadas fazem a classificação dos oradores e o respetivo reconhecimento da fala de uma forma supervisionada, onde a pyAudioAnalysis é a única que consegue este tipo de tarefas, tanto de forma supervisionada como de forma não-supervisionada. Estas operações não se aplicam na Pydub, pelo facto de esta fazer apenas a manipulação do áudio.

Tipo de Ficheiros Suportados

A ALIZE, a Pydub e a pyAudioAnalysis suportam praticamente todos os tipos de ficheiros de áudio existentes. Quanto à UIS-RNN, esta trabalha apenas com ficheiros NPZ, referentes à NumPy, que é uma biblioteca existente no Python.

Integração como sendo uma API ou Micro-Serviço

A UIS-RNN tem uma arquitetura que permite apenas a sua utilização num ambiente local, no qual são necessários recursos para fazer o seu processamento intensivo. Em consequência disso, esta biblioteca não é o ideal para integrar numa aplicação *web*, como se de uma API ou micro-serviço se tratasse. As restantes tecnologias enquadram-se neste tipo de arquitetura, contudo, a ALIZE é mais apropriada para dispositivos *mobile*, através da camada *Android-ALIZE*.

	ALIZE	**UIS-RNN**	**Pydub**	**pyAudioAnalysis**
Linguagem	C++	Python	Python	Python
Funcionalidades Principais	Reconhecimento da Fala	Segmentação de Áudio	Manipulação de Áudio	Análise e Segmentação de Áudio
Reconhecimento da Fala	Sim	Sim	Não	Sim
Segmentação de Áudio	Sim	Sim	Não	Sim
Tipo de Classificação	Supervisionada	Supervisionada	-	Supervisionada e Não-Supervisionada
Tipo de Ficheiros Suportados	Vários	NPZ	Vários	Vários
API ou Micro-Serviço	Sim	Não	Sim	Sim

Tabela 2.1: Tabela Comparativa entre a ALIZE, UIS-RNN, Pydub e pyAudioAnalysis

2.4 Medidas de Avaliação para o *Speech-to-Text*

Para além das medidas de avaliação convencionais como a precisão, a *recall*, a *f-measure*, entre outras, o módulo STT pode ser avaliado na qualidade da transcrição, feita pelo seu reconhecedor de fala. Assim, a presente secção apresenta algumas medidas que podem ser utilizadas para avaliar o seu desempenho.

2.4.1 *Diarization Error Rate*

O *Diarization Error Rate (DER)* é a medida de avaliação utilizada nos sistemas que suportam SD, tendo como objetivo o cálculo da percentagem da taxa de erro dos mesmos. A implementação desta medida é feita em dois passos: o primeiro passo consiste em fazer um mapeamento entre os rótulos associados a cada orador e as entidades encontradas em referência na conversa, enquanto o segundo passo consiste em calcular a taxa de erro da etapa anterior [12]. Para realizar este cálculo, são contabilizados os seguintes tipos de erro:

- **Erro do Orador**: percentagem de tempo em que é feita a classificação errada do orador;
- **Falso Alarme**: percentagem de tempo em que um hipotético orador é associado a um segmento de não-fala;
- **Discurso Perdido**: percentagem de tempo em que o segmento de fala válido não é detetado ou é compreendido como um segmento de não-fala;
- **Sobreposição do Orador**: percentagem de tempo onde várias pessoas, inseridas num determinado segmento de fala, não são atribuídas a nenhum orador.

Assim, a taxa de erro da *Speaker Diarization* é compreendida da seguinte forma:

$$\text{DER} = E_{Spkr} + E_{FA} + E_{MS} + E_{Ovl} \tag{2.1}$$

onde: E_{Spkr} = Erro do Orador
E_{FA} = Falso Alarme
E_{MS} = Discurso Perdido
E_{Ovl} = Sobreposição do Orador

2.4.2 Semelhança do Cosseno

A semelhança do cosseno é uma medida de avaliação encarregue de calcular o grau de semelhança entre duas fontes de texto, através do cosseno do ângulo de dois vetores. Para utilizar esta medida, ambos os textos analisados são convertidos em vetores, através da *Term Frequency - Inverse Document Frequency (tf-idf)* [23], que representa a normalização da frequência de cada termo em cada documento. Assim, a semelhança do cosseno é compreendida da seguinte forma:

$$cos(\Theta) = \frac{A.B}{\|A\| \, \|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2.2}$$

onde: A, B = Documentos
i = Termo do documento

A semelhança do cosseno lida com o tamanho total dos vetores de palavras, sendo ideal para situações em que é importante a frequência da repetição dos termos.

2.4.3 Coeficiente de Jaccard

O coeficiente de Jaccard é medida de avaliação, definida como sendo o tamanho da interseção de dois conjuntos, dividido pelo tamanho da sua união [23]. Perante dois conjuntos de palavras A e B, a semelhança de texto através do coeficiente de Jaccard é vista da seguinte forma:

$$jaccard(A, B) = \frac{A \cap B}{A \cup B} \tag{2.3}$$

O coeficiente de Jaccard lida apenas com um conjunto de palavras não repetidas, sendo ideal para casos em que a frequência dos termos não é importante.

CAPÍTULO 3

Análise de Texto

O presente capítulo introduz o conceito de análise de texto, apresentando os vários temas onde esta se enquadra e as tecnologias que são frequentemente utilizadas nos dias de hoje. Primeiramente, são descritas as várias áreas como o processamento, a compreensão e geração da linguagem natural e o reconhecimento de entidades, assim como o *Machine Learning* e o *Deep Learning*, seguidas da apresentação de várias tecnologias dedicadas à análise de texto. Por último, é definido o conceito de agente conversacional, sendo feito um levantamento das ferramentas dedicadas à sua construção.

3.1 Conceitos sobre a Análise de Texto

O texto é o tipo de informação mais utilizada atualmente. Por dia, são enviados centenas de milhões de dados, através de fontes como *e-mails*, páginas *web*, documentos, artigos, redes sociais, entre outras [24]. Dada a grande quantidade gerada, e também pelo facto de ser uma fonte de informação não estruturada, torna-se impossível a pesquisa, exploração e manipulação manual, feita em tempo útil, de forma a que seja relevante para uma pessoa ou organização [25].

Para combater tais problemas, surge a necessidade de criar sistemas de processamento de texto que consigam analisar grandes quantidades de dados, economizando tempo, custos e recursos humanos. Assim, estes sistemas de informação conseguem oferecer três capacidades distintas, porém interligadas entre si [26]:

- **Acesso à informação**: capacidade de retornar informação útil ao utilizador, quando este necessita. Com esta funcionalidade, é possível conectar a informação certa e relevante em tempo útil;
- **Aquisição de conhecimento**: capacidade que permite ao utilizador adquirir informação útil e codificada, através da análise de uma grande quantidade de dados, de forma a descobrir padrões existentes no texto submetido e criar nova informação ou conhecimento;

- **Organização de texto**: capacidade que permite anotar a coleção de documentos de texto em estruturas e tópicos com significado relevante, de tal forma a que a informação possa estar conectada e um utilizador possa navegar perante a mesma, ao seguir as suas estruturas. A organização do texto tem o papel de criar o acesso à informação e à aquisição de conhecimento de uma forma mais eficaz.

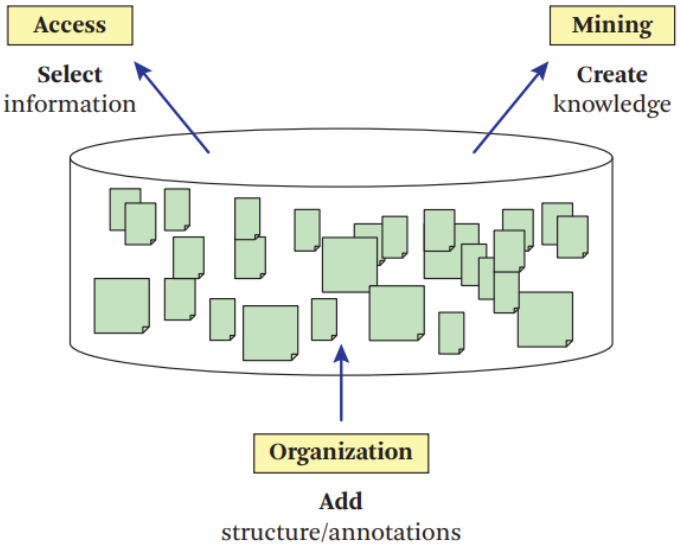

Figura 3.1: Representação dos Sistemas de Informação de texto [26]

A análise de texto é a área que estuda como aplicar métodos computacionalmente inteligentes, capazes de extrair e classificar automaticamente informação estruturada através de documentos, frases ou termos [25]. Por outras palavras, analisar texto é um processo autómato, que abrange a utilização de alguns dos seguintes métodos [27]:

- **Frequência das Palavras**: a frequência das palavras pode ser utilizada para listar as palavras ou conceitos mais utilizados num determinado texto. Este método pode ser útil para, por exemplo, analisar as expressões mais comuns de um utilizador, durante uma conversa;
- **Colocação**: a colocação ajuda a identificar as palavras que co-ocorrem, ou seja, é verificada a frequência com que dois ou mais termos aparecem sequencialmente;
- **Concordância**: a concordância ajuda a identificar o contexto e as instâncias de um conjunto de palavras, de forma a tentar descodificar algumas ambiguidades da linguagem humana;
- **Classificação de Texto**: a classificação de texto é o processo de atribuir marcadores ou categorias pré-definidas a texto não estruturado. É considerada uma das técnicas mais úteis no processamento de linguagem natural, pelo facto de conseguir organizar, estruturar e categorizar os dados;

- **Análise Sentimental**: a comunicação entre os humanos pode ser acompanhada de emoções. Em diálogos categorizados, como por exemplo, opiniões, o texto pode ser classificado como positivo, negativo ou neutro;
- **Análise de Tópicos**: a análise de tópicos é encarregue de perceber qual o tema associado a um determinado texto. Normalmente, é utilizado para estruturar e organizar os dados;
- **Deteção da Língua**: a deteção da língua permite a classificação do idioma falado, de acordo com o tipo de linguagem utilizada;
- **Deteção de Intenções**: os classificadores de texto também podem ser utilizados para detetar automaticamente qual a proposta ou intenção/ação de um determinado texto;
- **Extração de Texto**: a extração de texto consiste em extrair termos como tópicos, categorias e palavras-chave. Normalmente, este método é usado simultaneamente com a classificação de texto, de forma a detetar automaticamente a informação relevante;
- **Reconhecimento de Entidades**: um reconhecedor de entidades tem a capacidade de encontrar palavras, associadas a categorias como pessoas, localizações, empresas, entre outros;
- **Geração de Sumários**: gerar um sumário permite a criação de uma síntese, associada a um texto com uma extensão relativamente longa, desde que este continue a ser relevante, em comparação com a fonte de informação original;
- **Desambiguação de Palavras**: existem várias palavras que podem conter mais do que um significado, em contextos diferentes. A desambiguação destes mesmos termos trata de identificar o seu sentido, com o intuito de corrigir a sua incorreta classificação;
- ***Clustering***: o *clustering* tem o objetivo de agrupar vários dados não estruturados em diversos conjuntos, mediante o grau de semelhança de uma ou várias propriedades.

Assim, a análise de texto é um processo escalável, robusto o suficiente para criar relatórios consistentes em tempo real, ao contrário do comportamento humano [27], e abrangente pelo facto de poder ser aplicada em vários domínios como a Educação [28], Química [29], Biologia, Redes Sociais [25][30], Medicina, contexto empresarial e científico, ou até áreas de âmbito Biomédico [31][32].

3.2 Processamento de Linguagem Natural

O processamento de linguagem natural (NLP - *Natural Language Processing*), é um ramo da Inteligência Artificial (IA) que foca-se em agilizar a comunicação entre as máquinas e os humanos, através da linguagem natural humana. O principal objetivo desta metodologia consiste em interpretar, compreender e manipular as palavras utilizadas pelos seus utilizadores.

O NLP inclui várias técnicas de processamento e análise, como técnicas baseadas em *Machine Learning* ou métodos baseados em regras e algoritmia, sendo possível realizar algumas das seguintes operações [33]:

- ***Lemmatization***: redução das várias formas flexionadas para uma só, tendo em conta a análise morfológica da mesma;
- **Segmentação Morfológica**: divisão de palavras em unidades individuais, chamadas de morfemas;
- **Segmentação de palavras**: divisão de uma grande quantidade de texto em unidades distintas. Normalmente, esta funcionalidade é também chamada de *tokenization*;
- ***Part-of-Speech (POS) Tagging***: identificação de cada palavra como sendo um pronome, determinante, adjetivo ou nome;
- ***Parsing***: realização da análise gramatical para cada frase;
- **Segmentação de Frases**: criação de limites entre as frases, dentro de uma vasta quantidade de texto;
- ***Stemming***: exclusão do início e/ou do fim de cada palavra, tendo em conta uma lista de prefixos e sufixos, que podem ser encontrados numa lista flexionada.

3.3 Compreensão da Linguagem Natural

A compreensão da linguagem natural (NLU - *Natural Language Understanding*), é uma categoria do NLP que transforma a linguagem humana em informação estruturada, de modo a que a máquina consiga compreender e consequentemente agir. Para isso, é necessário converter o texto em ontologias, utilizando modelos estatísticos e combinações de regras. As entidades existentes no texto devem de ser extraídas e identificadas, de modo a que haja um significado semântico num contexto associado [34].

A NLU lida essencialmente com três termos: intenção (*intent*), entidade (*entity*) e frases de treino (*training phrases*). As intenções são classes que capturam a comunicação principal de um texto, podendo ser associadas a um tópico ou ação, como por exemplo, pesquisar por um restaurante, perguntar sobre a meteorologia, comprar bilhetes para um espetáculo, entre outros. Por outro lado, uma entidade é uma informação chave, contida numa intenção, podendo representar diversas categorias como restaurantes, locais, pessoas, empresas, etc. Quanto às frase de treino, as mesmas são um conjunto de frases suportadas que contém anotações de entidades e que por sua vez, estão enquadradas numa intenção. Estas frases podem ser aprendidas através do treino do modelo linguístico, extraindo posteriormente as informações associadas. A Figura 3.2 apresenta um exemplo de configuração de uma intenção, com uma entidade e frases de treino associadas.

```
{
  "text": "show me chinese restaurants",
  "intent": "restaurant_search",
  "entities": [
    {
      "start": 8,
      "end": 15,
      "value": "chinese",
      "entity": "cuisine"
    }
  ]
}
```


Figura 3.2: Exemplo de configuração de uma intenção [35]

3.4 Geração de Linguagem Natural

A geração de linguagem natural (NLG - *Natural Language Generation*) é uma das áreas da IA que preocupa-se em construir, através dos dados estruturados, sistemas capazes de produzir respostas compreendidas pela linguagem humana. Estes resultados são, na maioria das vezes, texto. Contudo, é possível haver a personalização das respostas, incluindo elementos multimédia como imagens e vídeo [36].

Tradicionalmente, a implementação desta metodologia tem em conta vários pormenores, nomeadamente determinar qual a informação a incluir no texto, qual a ordem de apresentação da resposta, quais as informações a apresentar, no caso de ser gerada uma única frase, quais as palavras mais indicadas para expressar corretamente a informação e identificar o domínio do problema em causa, assim como a criação de frases que, não só façam sentido, como também acrescentem valor para o utilizador [36].

A NLG é algo que está relacionado com o NLP, apesar de terem funções completamente distintas. Enquanto o NLP recebe informação não estruturada através do texto emitido pelos seres humanos, a NLG recorre a dados já devidamente organizados [37], realizando tarefas como a criação de relatórios de análise e estatísticos, a geração de descrições para um produto em particular, ou até fazer publicações em *blogs*, o que resulta numa percentagem de intervenção humana bastante reduzida [38].

3.5 Reconhecimento de Entidades

O reconhecimento de entidades (NER - *Named Entity Recognition*), é o método de localizar, extrair e identificar elementos textuais em categorias pré-definidas como locais, pessoas, organizações, objetos, quantidades, entre outras [39]. Este método pode ser aplicado em três modelos distintos: modelos baseados em regras, modelos baseados apenas em *Machine*

Learning e modelos híbridos, que combinam as duas abordagens anteriores. O modelo baseado em regras é uma técnica simples e que apresenta ótimos resultados. Contudo, esta requer um processo manual intensivo para lidar com as regras de identificação, para além dos esforços adicionais elevados quando são feitas mudanças acentuadas no domínio do problema. Por outro lado, o modelo baseado em *Machine Learning*, apesar de não conseguir alcançar resultados tão eficientes, requer pouco esforço manual, especialmente na integração de novos conteúdos [24].

Grande parte das técnicas estudadas para o NER optam por utilizar o modelo de classificação híbrido, reduzindo assim o esforço manual, a par da implementação de algumas regras específicas que permitem alcançar melhores resultados. Aplicado este modelo, a grande diferença entre as técnicas que o adotam incide sobre os algoritmos de classificação utilizados. Por exemplo, Chiarello [39] utiliza o classificador binário *Support Vector Machine (SVM)* no treino da extração das entidades, enquanto Sheikhshab [31] e Leaman [32] utilizam o *Conditional Random Field (CRF)*, que é um modelo utilizado para prever sequências de texto, mediante a informação obtida das associações anteriores. Por outro lado, Singhal [40] utiliza o *AdaBoostMH*, que é uma versão estendida do *AdaBoost*, sendo um algoritmo meta-heurístico, encarregue de melhorar a *performance* dos algoritmos tradicionais. Existem também abordagens que utilizam as *Hidden Markov Models (HMM)*, de forma a calcular as probabilidades condicionadas de retornar uma determinada sequência, consoante a sequência inicial fornecida, ou até a desambiguação entre entidades, mediante a utilização do *Pair-Linking*, como fazem Phan e Sun [41]. Por último, há quem utilize vários modelos de classificação e depois avalie os resultados obtidos, escolhendo no final aquele que alcançou valores mais satisfatórios. Esta situação ocorre na teoria criada por Akhtiamov [42] e por Korvigo [29], onde este último faz uma combinação entre as redes neuronais *Convolutional Neural Networks (CNN)* e *Recurrent Neural Networks (RNN).*

Apesar da maioria dos sistemas de criação e reconhecimento de entidades anotadas serem desenvolvidos com algoritmos de classificação, existem outras abordagens. O caso da teoria concebida por Raposo [43] é um exemplo da utilização das técnicas de NLP, a par da combinação de várias métricas da área da Recuperação de Informação, que permitem tratar e desambiguar o texto, obtendo no fim um conjunto de resultados relevantes, mediante o tipo de classificação implementada.

3.6 *Machine Learning*

O *Machine Learning (ML)* é uma subárea da Inteligência Artificial que tem o objetivo de fazer com que as máquinas aprendam de forma autónoma. Nesta área, é explorada e estudada a construção de algoritmos que tenham a capacidade de reaprender através dos seus erros e de detetar padrões existentes nos dados observados, de forma a construir modelos de previsão. Por outras palavras, um modelo de previsão é um conjunto de regras e procedimentos que permite aprender através de um conjunto de amostras, generalizando depois para as novas

instâncias, sem ter quaisquer necessidades de criar regras explicitamente pré-programadas [44].

De acordo com os dados de entrada, os algoritmos de ML são classificados em três categorias [44]:

- **Aprendizagem Supervisionada**: é apresentada uma relação entre os dados de entrada e o resultado final, através de um conjunto de treino. O objetivo consiste em aprender a regra de mapeamento, prevendo o resultado final dos novos dados;
- **Aprendizagem Não-Supervisionada**: não é fornecido nenhum conjunto de treino prévio. Com base nos dados de entrada, o objetivo consiste em descobrir padrões ou relações existentes nos mesmos;
- ***Reinforcement Learning***: interação com um ambiente dinâmico, onde é necessário alcançar um objetivo. Na sua execução, o agente aprende através de premiações e punições, conforme o seu comportamento, com o objetivo de maximizar a recompensa a longo prazo.

Os algoritmos de ML são ainda categorizados de outra forma, com base no seu *output* [45]:

- **Classificação**: associação de uma nova instância a uma classe já conhecida. Por outras palavras, é o processo de atribuir um rótulo a uma nova entrada;
- **Regressão**: algoritmo que é utilizado quando um *output* é contínuo, normalmente associado a valores numéricos ou categóricos;
- ***Clustering***: divisão dos dados em grupos, mediante a semelhança de uma determinada propriedade.

O ML é uma das áreas mais importantes na classificação do texto. A sua aplicação consiste em melhorar, acelerar e automatizar as funções subjacentes da análise de texto, assim como as tarefas presentes no NLP, transformando o texto não estruturado em dados utilizáveis e relevantes. Ao contrário de alguns algoritmos de programação baseados em regras, os modelos de ML conseguem lidar com novos dados, utilizando o seu conhecimento prévio para avaliar o caso. Estes modelos têm o objetivo de melhorar continuamente um conjunto de técnicas estatísticas que identificam partes de um discurso, entidades, análise sentimental, entre outros aspetos, através da sua aplicação a novos casos com a aprendizagem supervisionada, ou então através da extração de conhecimento com a utilização da aprendizagem não-supervisionada [46].

3.6.1 *Deep Learning*

O *Deep Learning (DL)* é uma categoria de algoritmos do ML, inspirados na estrutura e funcionamento de um cérebro, chamada de redes neuronais. Estes algoritmos trabalham com um modelo multi-camada que contém, no mínimo, três camadas. Neste conceito existe uma etapa de *feedforward*, onde cada camada aceita a informação transmitida pela camada anterior,

havendo depois a transferência de informação para a próxima, e assim consecutivamente até à última camada existente [47].

Uma rede neuronal depende de três aspetos fundamentais: os seus *inputs* e funções de ativação, a arquitetura da rede e o peso da conexão de cada *input*, chamados de *weights*. Estes parâmetros, assim como a atribuição de um valor ao *bias*, são fundamentais na obtenção do modelo final, onde a estratégia de inicialização deve de ser selecionada de acordo com a função de ativação, que é encarregue de definir o valor de saída do neurónio [44].

No DL existe também a etapa de *backpropagation*, que tem o objetivo de minimizar o valor da função de perda, onde esta quantifica a diferença entre o valor previsto e o valor real. Após o treino do modelo, os parâmetros são consequentemente atualizados, de modo a minimizar o valor da função anterior. Estes modelos costumam alcançar bons resultados, especialmente no processamento de grandes quantidades de dados, onde a extração de características é aprendida e realizada pela máquina, algo que não acontece nos algoritmos tradicionais de ML [47].

Dada a sua adaptabilidade, escalabilidade e eficiência, o DL é bastante usado na área do NLP, sendo aplicado em várias tarefas, como por exemplo, o *parsing* de dependências, a análise sentimental, o cálculo da semelhança contextual entre dois textos e a criação de sistemas pergunta-resposta.

3.7 Tecnologias de Análise de Texto

Nesta secção são apresentadas algumas tecnologias que permitem fazer o processamento e compreensão da linguagem natural humana.

3.7.1 GATE

A *General Architecture for Text Engineering (GATE)* é uma ferramenta *open-source*, capaz de desenvolver componentes de *software* relacionados com a manipulação da linguagem humana [48]. Tendo sido criada sensivelmente há duas décadas, esta *framework* continua a ser uma das mais usadas no contexto industrial e universitário, desde grandes empresas até pequenas *startups*, com orçamentos de milhões ou simplesmente projetos de âmbito académico.

A GATE é desenvolvida de forma comunitária, englobando um ecossistema de programadores, engenheiros de linguagens e investigadores de diversas áreas, espalhados por toda a parte. Assim, e com a evolução desde a data da sua criação, foram criados diversos componentes [49]:

- **GATE *Developer***: ambiente de desenvolvimento integrado para os componentes do processamento de linguagem, empacotados com um sistema de extração de informação e um conjunto de vários *plugins*;
- **GATE *Cloud***: hospedagem do processamento de texto em larga escala;

- **GATE *Teamware***: aplicação *web* de ambiente colaborativo para os projetos relacionados com a anotação semântica, construídos em torno de um fluxo de execução e de uma estrutura de serviços *back-end* otimizada;
- **GATE *Mimir***: servidor de indexação que pode ser utilizado para indexar e procurar sobre texto, anotações, ontologias e metadados semânticos;
- **GATE *Embedded***: biblioteca de objetos otimizada que pode ser incluída em diversas aplicações, dando acesso a todos os serviços utilizados no GATE *Developer* e outros;
- Arquitetura de alto nível;
- Processo para a criação de serviços robustos e sustentáveis.

Suportando várias técnicas do NLP como *parsing*, morfologia, *POS tagging* e ferramentas de recuperação e extração de informação [48] para dezenas línguas que não são portuguesas, entre outras tarefas, a GATE tem o objetivo de minimizar tempo e esforço através do desenvolvimento e manutenção de sistemas de recolha de informação. Um desses sistemas é o ANNIE. O ANNIE NER é um serviço *web* que oferece um conjunto de tarefas num só módulo, de forma a ser utilizado na descoberta das entidades. Para tal, são feitas pesquisas com base nos *gazetteers*, dicionários geográficos que contém estatísticas sociais e características físicas de cada local, e em máquinas de estado finito para a extração e classificação das palavras-chave, conseguindo identificar categorias como pessoas, locais, organizações, datas, endereços, entre outros tipos [24].

3.7.2 Stanford NER

A Stanford NER é uma biblioteca, construída em Java, que aplica o paradigma do NER. Desenvolvida na Universidade de Stanford, a mesma encontra-se embutida num conjunto, chamado de Stanford CoreNLP que contém técnicas de NLP como a *tokenization*, *POS tagging*, NER, *parsing*, desambiguação e co-referências. Esta biblioteca utiliza uma implementação geral da cadeia linear dos modelos CRF, isto é, ao treinar os próprios modelos com os dados anotados, é possível utilizar os mesmos para construir sequências no NER. Para treinar este modelos, é feito um cruzamento de dados entre vários *corpus*, onde são extraídas várias características como a ortografia, a extração de prefixos e sufixos e a criação de conjunções, durante a fase do pré-processamento.

A Stanford NER tem a vantagem de ser uma ferramenta livre e de domínio adaptável: o utilizador conseguir definir *gazetteers* próprios, assim como personalizar e treinar modelos, além dos que são inicialmente fornecidos. Apesar da sua flexibilidade e capacidade de personalização, esta ferramenta apresenta uma complexidade temporal elevada no CRF, juntamente com um número de entidades relativamente pequeno [24]. A Tabela 3.1 apresenta o tipo de entidades disponibilizadas em cada modelo.

Modelo	Tipo de Entidades
3 Classes	Locais, Pessoas e Organizações
4 Classes	Locais, Pessoas, Organizações e Entidades Gerais
7 Classes	Locais, Pessoas, Organizações, Dinheiro, Percentagens, Datas e Tempo

Tabela 3.1: Modelos disponíveis na Stanford NER [50]

3.7.3 Open Calais

O Open Calais é um serviço *web*, utilizado para incorporar funcionalidades semânticas em vários contextos como sistemas de gestão, aplicações e páginas *web*, através da análise de texto. Esta ferramenta, através da utilização de métodos estatísticos, ML e de métodos baseados em padrões, contém um mecanismo de NLP capaz de localizar entidades, factos e eventos, assim como relações entre os mesmos. Para tal, é utilizado um sistema baseado na construção de regras específicas, onde é feita uma combinação entre as regras descobertas e o campo lexical associado [51]. Desta forma, é possível extrair mais de 30 tipos de entidades desde nomes de pessoas, cidades, países, estados/distritos, organizações, categorias gerais, negócios, fármacos, entre outros, suportados em Inglês, Francês e Espanhol.

Neste serviço, existe um processamento dedicado a entidades mais eminentes como nomes de pessoas e organizações. Para as entidades deste tipo, onde a taxa de ambiguidade é maior, é feito um processo de desambiguação, que consiste na resolução de co-referências, na implementação de uma validação cruzada com os conjuntos de dados disponibilizados e na normalização do texto. A desambiguação é feita com o auxílio do Linked Open Data, que é um sistema agregador de dados *web*, interligados entre si, através de regras sofisticadas para a descoberta de entidades, onde são criadas relações entre os termos, e consequentemente, associações entre estes e as categorias existentes [24].

O Open Calais é um analisador de texto bastante popular, sendo conhecido por gerar ficheiros XML que contém, no formato de *Resource Description Framework (RDF)*, os metadados semânticos referentes aos termos extraídos, onde constam atributos como o id, nome e tipo para cada um dos anteriores [52].

3.7.4 MeaningCloud

O MeaningCloud, anteriormente conhecido como Textalytics, é um serviço que permite aos utilizadores fazerem a análise de texto e criarem relações semânticas em qualquer aplicação ou sistema. Relançado em 2015, este serviço suporta uma nova versão que tem como foco a análise da voz e o *feedback* dado pelo cliente, a análise do conteúdo das redes sociais, a gestão, codificação e classificação de documentos e a produção de novo conteúdo relevante. Baseado no modelo de reconhecimento híbrido, através da combinação de um sistema baseado em regras com algoritmos de ML, este *software* utiliza coleções de dados de treino manualmente anotadas para construir modelos de deteção e classificação de entidades. Todos os modelos de classificação possuem uma ontologia que contém à volta de 200 categorias diferentes, onde

é também feita a análise co-referencial e a desambiguação dos termos, através de recursos externos como a Wikipédia [24]. A próxima figura demonstra o fluxo de treino e classificação feito nos modelos do MeaningCloud.

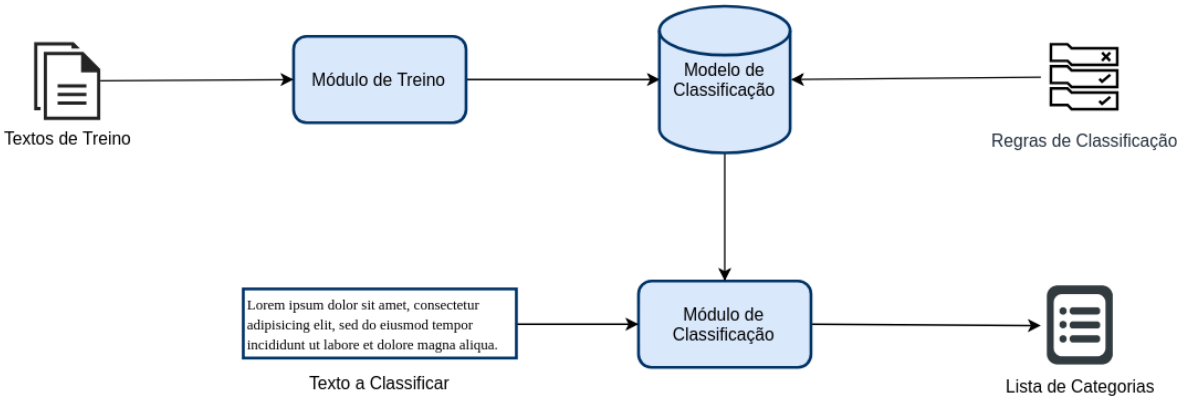

Figura 3.3: Fluxo de treino e classificação do MeaningCloud [24]

Suportando línguas como o Inglês, Espanhol, Português, Francês e Italiano, o MeaningCloud é poderoso, adaptável e bastante flexível, onde é possível construir modelos e *gazetteers* próprios, personalizar o sistema através de interfaces gráficas, usufruir da análise sentimental, integrar os dados com outras ferramentas como o Microsoft Excel e fazer a extração de características, sem ser necessário qualquer tipo de instalação ou desenvolvimento. É uma plataforma fácil de se aprender e utilizar, estando também disponível na GATE, através da instalação de um *plugin* [53].

3.7.5 Apache OpenNLP

A Apache OpenNLP é uma biblioteca que contém um conjunto de ferramentas, baseadas em ML, para o processamento de texto, codificado em linguagem natural. Esta suporta as mais diversas tarefas de NLP, tal como *tokenization*, segmentação de frases, POS *tagging*, reconhecimento de entidades, *chunking*, *parsing* e resolução de co-referências [24].

Para a construção dos modelos linguísticos, a OpenNLP baseia-se nas abordagens de ML da entropia máxima e na rede neuronal artificial *perceptron*, que é um classificador binário de uma só camada. Desta forma, a biblioteca tem o objetivo de criar um único módulo que suporte todas as tarefas do processamento de texto, fornecendo modelos pré-construídos em várias línguas, bem como os recursos de texto anotados, dos quais estes são derivados. Cada modelo pode ser treinado através de vários conjuntos de dados disponíveis e consequentemente avaliados, sendo também possível alterar o seu conteúdo [54].

Um dos pontos fortes presentes nesta biblioteca são a flexibilidade, adaptabilidade e capacidade do utilizador conseguir personalizar o conteúdo dos seus modelos de extração. Para além disso, a OpenNLP apresenta outras funcionalidades extra como sumariar parágrafos, artigos ou documentos, pesquisar por palavras ou sinónimos, mesmo que estas se encontrem alteradas ou incompletas, traduzir de um língua para outra, analisar o *feedback* recebido, reconhecer

a fala e construir relatórios de informação [55]. Por outro lado, existem algumas lacunas que necessitam de ser corrigidas, nomeadamente a implementação da análise sentimental das opiniões dadas, a modelação de tópicos e categorias e a desambiguação contextual, que é fundamental para a separação de termos e conceitos [24].

3.7.6 TextRazor

A TextRazor é uma ferramenta, fornecida via API, que tem a capacidade de processar texto proveniente de fontes como *e-mails*, *tweets*, documentos e pesquisas. Com a capacidade de analisar texto para línguas como o Inglês, Chinês, Holandês, Francês, Alemão, Italiano, Japonês, Polaco, Português, Russo, Espanhol e Sueco, são oferecidas as seguintes funcionalidades [56]:

- Identificação e extração de entidades;
- Criação e Desambiguação de relações;
- Classificação automática de tópicos;
- Extração de palavras-chave;
- Extração de relações e dependências;
- Construção de regras específicas do domínio do problema;
- Deteção de 142 línguas.

No que toca à identificação de entidades e palavras-chave, é utilizado um modelo de extração baseado numa abordagem híbrida, ou seja, é feita uma combinação entre modelos baseados em ML e em regras. A abordagem estatística do ML foca-se em extrair entidades que contenham caracteres especiais ou números, como é o caso das *hashtags*, dos *e-mails* e dos números de telefone, sendo esta metodologia integrada num mecanismo de correspondência, construído através de regras específicas. O método baseado em regras recolhe um vasto conjunto de dados através de fontes como a Wikipédia, a DBpedia e o Wikidata, conseguindo gerar dicionários com diversas combinações de entidades. Com a criação de largas centenas de entidades distintas, é possível fazer a desambiguação dos termos semelhantes. Para tal, são analisados os contextos envolventes do documento, onde estes são combinados com as bases de conhecimento dos factos. Assim, são aplicados diversos métodos, desde métodos superficiais, métodos baseados em grafos e métodos linguísticos aprofundados que permitem atribuir um valor de confiança a cada entidade, mediante o contexto em que se encontra [24].

A TextRazor consegue interpretar pesquisas de homónimos, sinónimos, siglas e palavras traduzidas e até incompletas, para além de permitir identificar pessoas, locais e organizações jamais anteriormente mencionadas, assim como a implementação de novas expressões regulares [56]. Apesar disso, as suas capacidades de adaptação são limitadas às regras que foram construídas manualmente pelo utilizador, não sendo possível voltar a treinar ou adicionar um novo modelo [24].

3.7.7 NLTK

A *Natural Language Toolkit (NLTK)* é uma *framework* gratuita, multi-plataforma e desenvolvida em Python, que oferece um conjunto de algoritmos capazes de lidar com a linguagem

natural humana em diferentes contextos, desde a investigação, educação, engenharia e aplicações industriais. A mesma oferece um conjunto de interfaces para mais de 50 *corpus* e recursos lexicais, suportados em catorze línguas, incluindo o Português, e uma panóplia de ferramentas NLP como a *tokenization*, *stemming*, POS *tagging*, classificação, *parsing*, desambiguação, segmentação de frases, remoção de *stopwords*, reconhecimento de entidades e raciocínio semântico [57][58].

A NLTK é constituída pelos seguintes módulos independentes, onde cada um define a sua estrutura de dados ou tarefa em particular [59]:

- **Módulo *Token***: fornece classes básicas para o processamento individual de elementos textuais, tal como palavras ou frases;
- **Módulo *Tree***: define estruturas de dados como árvores sintáticas e morfológicas para serem representadas sobre palavras ou frases;
- **Módulo da Probabilidade**: implementa classes que codificam as distribuições da frequência e da probabilidade, incluindo uma variedade de técnicas estatísticas;
- **Módulo de *Parsing***: define uma interface de alto nível para produzir árvores que representam as estruturas dos textos;
- **Módulo de *Tagging***: define uma interface para acrescentar informações suplementares a cada *token* do texto;
- **Modelos de Autómatos de Estado Finito**: definem o tipo de dados para codificar um autómato de estado finito, assim como uma interface para criar autómatos através de expressões regulares;
- ***Type Checking***: o tempo gasto a fazer *debugging* é um fator importante na facilidade de uso da ferramenta. Para reduzir este valor, é fornecido um módulo que pode ser usado para assegurar que determinadas funções estão a fornecer os argumentos esperados;
- **Visualização**: define interfaces gráficas para visualizar e manipular estruturas de dados, assim como ferramentas para experimentar com as tarefas de NLP;
- **Classificação de Texto**: define uma interface para classificar textos em determinadas categorias. Este módulo é implementado por outros dois: um módulo que define um classificador de texto, com base nas redes *Bayesianas* e outro que faz o mesmo com a entropia máxima.

Para além da modularidade presente, a NLTK é uma ferramenta simples por dar um conhecimento prático do NLP aos utilizadores, consistente por apresentar um conjunto de interfaces e estruturas de dados regular, com nomes facilmente compreensíveis, extensível no sentido de facilitar a introdução de novos módulos ou de expandir os já existentes e explicativa pelo facto de estar bem documentada. Por outro lado, a mesma apresenta problemas de *performance* na realização de algumas tarefas, não contém vetores de palavras integrados e não fornece modelos de redes neuronais.

3.7.8 spaCy

A spaCy é uma biblioteca *open-source* e desenvolvida em Python, sendo especificamente desenhada para a sua utilização em produção, de forma a contribuir na construção de aplicações que processam e compreendem grandes quantidades de texto. Suportando características como a *tokenization*, POS *tagging*, *parsing* de dependências, *lemmatization*, NER, semelhança entre frases, classificação de texto, associações baseadas em regras e treino de modelos, esta biblioteca, para além de sistemas de NLP, consegue desenvolver sistemas de NLU ou de extração de informação, assim como pré-processar texto para o DL [60].

Enquanto alguns modelos linguísticos trabalham de forma independente, outros requerem modelos estatísticos para serem carregados, permitindo a previsão de anotações linguísticas. As anotações linguísticas oferecidas por esta biblioteca fornecem informações sobre a estrutura gramatical do texto, o que permite perceber, por exemplo, se uma determinada palavra corresponde a um verbo ou nome, mediante o seu contexto. A spaCy oferece modelos estatísticos para várias línguas, desde Inglês, Português, Alemão, Grego, Holandês, Francês, Espanhol e Italiano, podendo diferir no seu tamanho, velocidade, consumo da memória e precisão, conforme o caso de uso e o texto com o qual se está a trabalhar. Tipicamente, estes modelos contém os seguintes componentes [60]:

- Pesos binários para o POS *tagging*, *parsing* e identificador de entidades, de forma a prever as anotações no contexto;
- Entradas lexicais no vocabulário, ou seja, palavras e respetivos atributos;
- Vetores de palavras, ou seja, representações multi-dimensionais que permitem determinar o grau de semelhança entre duas ou mais palavras;
- Opções de configuração como as do *pipeline* e do processamento, de forma a que a biblioteca esteja no estado correto quando o modelo for carregado.

Quando o NLP é chamado para analisar o texto, a primeira coisa que a spaCy faz é a *tokenization*, seguida da construção de um objeto do tipo documento. O texto é processado em vários passos distintos, ao que se pode chamar de *pipeline*. Como apresentado na Figura 3.4, este *pipeline* contém, por defeito, um *tagger*, que é um classificador de um determinado excerto de texto, um *parser* e um reconhecedor de entidades. Cada um destes componentes retorna o documento processado, que é depois transferido para o componente seguinte. Este processo é dependente de um modelo estatístico e das suas capacidades, onde são tomadas decisões mediante a sua previsão, com base em exemplos de textos e associações que foram utilizados no treino do classificador. Todos estes elementos podem ter a intervenção do utilizador, nomeadamente a edição da lista de componentes presentes no *pipeline* e o *feedback* dado ao modelo, consoante a sua classificação gerada [61].

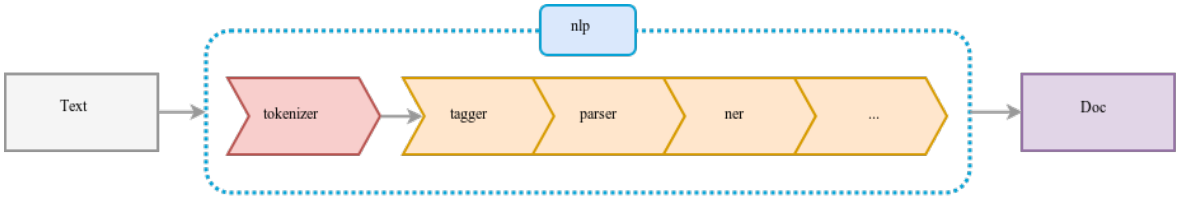

Figura 3.4: *Pipeline* da spaCy [60]

A spaCy é uma *framework* veloz, precisa e integrável com ferramentas terceiras, tendo também a capacidade de classificar cerca de vinte categorias de entidades, como por exemplo, pessoas, nacionalidades, grupos religiosos e políticos, datas, objetos, medidas, entre outras. Apesar de todo o seu potencial, é uma biblioteca que contém um processo de *tokenization* relativamente lento e que ainda não dá a flexibilidade suficiente aos seus utilizadores.

3.7.9 BERT

O *Bidirectional Encoder Representations from Transformers (BERT)* é um novo modelo linguístico *open-source*, capaz de analisar texto não estruturado e proveniente de várias fontes, em 103 línguas distintas. Contrariamente aos modelos de representação linguística atuais, este modelo é treinado de uma forma bidirecional, de modo a ter em conta o contexto e fluxo linguístico, algo que não acontece nos modelos de apenas uma direção [62].

O BERT utiliza um transformador que aprende relações contextuais entre palavras ou sub-palavras de um determinado texto. Este elemento contém dois mecanismos separados: um codificador que lê o texto recebido e um descodificador que produz uma previsão para a tarefa em causa. O codificador, contrariamente aos modelos tradicionais que leem o texto de forma sequencial (da esquerda para a direita ou vice-versa), é bidirecional, no sentido de ler uma sequência de texto inteira de uma vez só. Esta característica permite que o modelo consiga aprender o contexto de uma palavra, com base na sua periferia. A Figura 3.5 apresenta uma arquitetura de alto nível de um transformador, recebendo como *input* uma sequência de *tokens* incorporados em vetores e processados na rede neuronal, enquanto o *output* é uma sequência de vetores, todos do mesmo tamanho, em que cada um corresponde a um *token* de entrada com o mesmo índice [63].

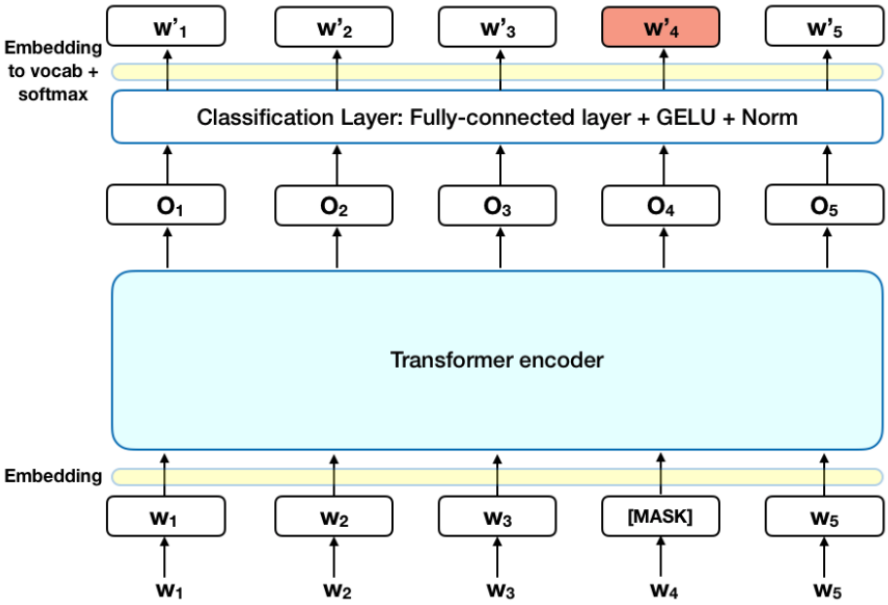

Figura 3.5: Transformador utilizado no BERT [63]

Quando os modelos linguísticos são treinados, existe o desafio de conseguir prever a próxima palavra, mediante a sequência do texto processada. De forma a ultrapassar a abordagem unidirecional e limitada deste tipo de operações, o BERT utiliza duas estratégias de treino: a *Masked Linguistic Model (MLM)* e a *Next Step Prediction (NSP)*. A MLM consiste em substituir, aproximadamente, 15% das palavras de cada frase pelo *token [MASK]*. Assim, o modelo tenta prever o valor original dos *tokens* que foram substituídos pelo *[MASK]*, com base no contexto fornecido pelas outras palavras da sequência que mantiveram-se intactas. Para prever quais os termos que foram modificados, é necessário executar os seguintes passos:

1. Adicionar uma camada de classificação no topo do *output* do classificador;
2. Multiplicar os vetores de *output* pela matriz embutida, transformando-os em dimensões de vocabulário;
3. Calcular a probabilidade de cada palavra pertencer ao vocabulário.

Durante o treino do BERT, são recebidos vários pares de frases, onde o modelo aprende a prever se, num determinado documento, uma frase é o par da frase anterior. Durante o processo de aprendizagem, cerca de 50% dos dados de entradas são pares de frases corretos, onde uma é a subsequente da outra. Nos restantes 50%, o processo é feito de forma diferente: a segunda frase é escolhida de forma completamente aleatória, através do *corpus* disponibilizado. Assim, e de forma a facilitar a identificação dos pares criados, os mesmos são processados da seguinte forma, antes de ser utilizado o modelo (Figura 3.6):

1. Em cada frase, são inseridos dois *tokens*: o *token* [CLS] (*classification*), inserido no seu início e que tem o objetivo de permitir realizar tarefas de classificação, e o [SEP] (*separator*), que é inserido no final, estando encarregue de fazer a separação entre a frase atual e a seguinte;

2. É feita a incorporação da frases, indicando se a frase A ou B é adicionada a cada *token*. As incorporações das frases são semelhantes ao conceito dos *tokens* incorporados, com um vocabulário de dois elementos (frases A e B);
3. É adicionada uma incorporação posicional, de forma a indicar a sua posição na sequência.

Assim, para prever se a segunda frase está conectada com a primeira, toda a sequência de texto tem de passar pelo modelo do transformador. O *output* dos *tokens* [CLS] é transformado num vetor, através da utilização de uma camada de classificação simples, e por último, é feito o cálculo da probabilidade de ser um par válido [63].

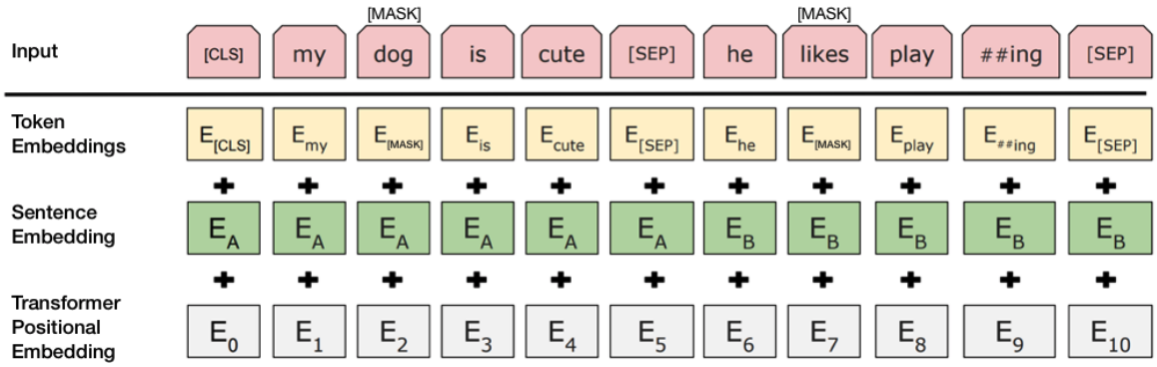

Figura 3.6: Representação do *input* no BERT [62]

O BERT pode ser utilizado em diversas tarefas, nomeadamente na classificação de texto, análise sentimental e aplicação de ferramentas NLP, tendo a capacidade de afinar os seus componentes pré-treinados, através da utilização de apenas uma camada adicional. Os seus modelos são atualmente uns dos maiores no mercado, tendo o seu modelo base cerca de 110 milhões de parâmetros, enquanto o completo ultrapassa os 345 milhões [63].

3.7.10 Comparativo

A Tabela 3.2 é referente à síntese das propriedades das tecnologias de análise de texto anteriormente apresentadas.

Número de Línguas

Todas as tecnologias analisadas conseguem compreender mais do que uma língua. Dentro deste lote, cujo destaque vai para o BERT pelo facto de suportar 103 idiomas diferentes, o Inglês, Francês, Espanhol e Italiano são as línguas mais compreendidas. Quanto ao Português, existe suporte em maioria dos casos, exceto na GATE, na Stanford NER e no Open Calais.

Número de Entidades

Grande parte das tecnologias consegue identificar um conjunto significativo de entidades. Apesar disso, para a língua portuguesa só é feita a identificação de categorias como pessoas, organizações e alguns locais, à exceção da Text Razor e do BERT que conseguem classificar um conjunto de termos na ordem das centenas e dos milhões, respetivamente.

Quanto à Apache OpenNLP e à NLTK, através da documentação de ambas não é possível compreender qual a agilidade da classificação do seu modelo de extração.

Modelo de Extração

Enquanto a GATE e a Open Calais seguem a abordagem baseada em regras, a Apache OpenNLP e a spaCy optam por um modelo de ML. Combinando as duas abordagens anteriores, o modelo híbrido é utilizado no MeaningCloud e na TextRazor.

Diferente dos três tipos de extração comuns, a Stanford NER usufrui do algoritmo CRF, enquanto o BERT utiliza um modelo que é treinado de uma forma bidirecional.

Desambiguação

Quase todas as tecnologias analisadas, com exceção da GATE e da Apache OpenNLP, têm a capacidade de fazer a desambiguação das palavras. Quanto à Stanford NER, apesar de não aplicar esta funcionalidade à priori, a mesma permite a instalação de ferramentas de suporte que realizam operações do género.

Open-Source

Quase todas as tecnologias são gratuitas ou até comunitárias, exceto o MeaningCloud e a TextRazor. O BERT, apesar de ser de acesso livre, contém um conjunto de dados de grandes proporções, sendo necessário a aquisição de *hardware* especializado em processamento, que custa vários milhares de euros.

Tecnologia	**Número de Línguas**	**Número de Entidades**	**Modelo de Extração**	**Desambiguação**	***Open-Source***
GATE	7	9	Baseado em Regras	Não	Sim
Stanford NER	4	3, 4 ou 7	CRF	Ferramentas de Suporte	Sim
Open Calais	3	39	Baseado em Regras	Sim	Sim
Meaning Cloud	6 (Português)	200	Modelo Híbrido	Sim	Não
Apache OpenNLP	7 (Português)	Não definido	Modelo de ML	Não	Sim
Text Razor	12 (Português)	Centenas	Modelo Híbrido	Sim	Não
NLTK	14 (Português)	Não definido	Baseado em Regras	Sim	Sim
spaCy	8 (Português)	20	Modelo de ML	Sim	Sim
BERT	103 (Português)	Milhões	Modelo Bidirecional	Sim	Sim

Tabela 3.2: Tabela Comparativa das Tecnologias de Análise de Texto

Para aplicar todo o conteúdo envolvente, desde a análise de texto até à resposta gerada ou tarefa realizada para o utilizador, é necessário construir um agente que suporte todas as funcionalidades do NLP, da NLU e da NLG, podendo também usar ML. Assim, e com a necessidade de criar algo capaz de comunicar com os seus clientes, através da linguagem natural, surgem os agentes conversacionais. Os agentes conversacionais são um conjunto de sistema autónomos que comunicam com os humanos por texto ou fala, através de várias plataformas como páginas *web*, aplicações e canais de comunicação [64].

Dependendo da forma como é programado, um agente conversacional pode dividir-se em duas categorias: agentes simples e agentes complexos. Os agentes simples trabalham com base na criação de comandos pré-escritos. Cada um destes comandos deve de ser especificamente implementado, através da utilização de expressões regulares ou de outras formas que permitam analisar o texto. Se estes agentes forem confrontados com perguntas que estão de acordo com as regras previamente estabelecidas, será gerada uma resposta, caso contrário, o pedido não será compreendido. Por outro lado, os agentes complexos não precisam de especificar qualquer tipo de regras. Através do ML, os mesmos conseguem compreender grande parte dos diálogos do utilizador, para além do seu modelo encontrar-se em constante melhoria, através da aprendizagem das conversas anteriormente realizadas. Apesar de ter um comportamento bastante satisfatório, um agente complexo requer bastante esforço inicial na sua elaboração e implementação [65].

Os agentes conversacionais, para além da capacidade de comunicar através de vários canais como o Facebook Messenger, Slack, Skype, Telegram, mensagens instantâneas, entre outros, conseguem fazer um variadíssimo conjunto de tarefas, desde tarefas relacionadas com o negócio, agendamento de compromissos, eventos e lembretes, recolha de informação sobre os seus utilizadores, apoio ao cliente e simplificação do processo de compra [65][62]. A sua capacidade em automatizar processos, gerando respostas em tempo útil, faz com que não seja necessário qualquer tipo de intervenção humana ou de custos adicionais, o que leva a um aumento de interesse por parte dos programadores e dos empreendedores.

3.8.1 Plataformas de Construção de Agentes

Com a intenção de desenvolver agentes conversacionais, capazes de terem um comportamento aproximado ao de um ser humano, surge a necessidade de utilizar plataformas que permitam o seu desenvolvimento e integração num ambiente propício à sua utilização.

Microsoft Bot Framework

A Microsoft Bot Framework é uma plataforma que, através do Azure Bot Service, oferece um ambiente integrado, criado especificamente para a construção, desenvolvimento, teste e gestão de agentes inteligentes, num único lugar. Através da sua estrutura modular e extensível, fornecida pelo SDK, pelas ferramentas, pelos *templates* e pelos serviços de IA, é possível

criar agentes que utilizam a fala, compreendem a linguagem natural e que respondem a determinadas perguntas, entre outras aplicações [66].

O Azure Bot Service e o Bot Framework, oferecem as seguintes funcionalidades [66]:

- Bot Framework SDK para desenvolver agentes;
- Bot Framework *Tools* para cobrir o fluxo de execução do agente em *end-to-end*;
- Bot Framework *Service* para enviar e receber mensagens e eventos entre os agentes e plataformas de comunicação como o Skype, Facebook Messenger e Slack;
- Implementação do agente e configuração dos canais de comunicação;
- Azure Cognitive *Services* para construir aplicações inteligentes;
- Azure Storage para soluções de armazenamento em *cloud*.

Para desenvolver um agente conversacional, o Azure Bot Service e o Bot Framework oferecem um conjunto integrado de serviços e ferramentas que ajudam a facilitar o processo. Este consiste em várias etapas, começando pelo planeamento e construção, seguidos da execução de testes. Depois, o mesmo agente é publicado no Azure, de modo a que fique disponível para outros utilizadores, prosseguindo para a sua conexão e avaliação [66]. A próxima figura apresenta o fluxo de execução da construção do agente, etapa a etapa.

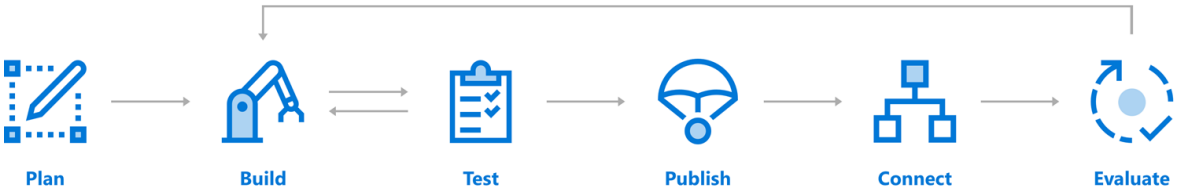

Figura 3.7: Etapas da criação de um agente na Microsoft Bot Framework [66]

Na Microsoft Bot Framework, um agente é visto como um serviço *web* que implementa uma interface de comunicação e que comunica com o Bot Framework Service para enviar e receber mensagens e eventos, com suporte para vários canais de comunicação. Assim, é possível criar agentes simples, em diversos ambientes e línguas, onde são oferecidos os seguintes componentes, de forma a estender as suas funcionalidades [66]:

- ***Natural Language Processing***: permite que o agente compreenda a linguagem natural, encontre erros ortográficos, use a fala e classifique intenções. Para tal, é necessário utilizar o *Language Understanding (LUIS)*, que é um serviço da Microsoft dedicado à NLU;
- **Resposta a perguntas**: existe uma base de conhecimento para responder, de uma forma mais natural, às questões do utilizador;
- **Gestão de vários modelos**: se estiver a ser utilizado mais do que um modelo, é determinado qual é o que se encontra em utilização, mediante o tipo de conversa;

- **Adicionar *cards* e botões**: aumento da experiência do utilizador com outros conteúdos que não sejam texto simples, como por exemplo, gráficos, menus e vídeos.

Dialogflow

A Dialogflow é uma plataforma, apoiada pela Google, que consiste em desenvolver tecnologias para as interações entre as máquinas e os humanos, com base nas conversações de linguagem natural [67]. Nesta *framework*, existe um conjunto de interfaces que podem ser baseadas em voz ou texto, tal como nas aplicações do reconhecimento de fala e nos agentes conversacionais. Desta forma, os utilizadores conseguem conectar-se a elementos como páginas *web*, aplicações móveis, assistentes virtuais, entre outros, onde é fornecido um mecanismo de NLU para processar e compreender os diálogos provenientes dos utilizadores [68].

O fluxo de conversação entre um utilizador e um dispositivo que usufrui da Dialogflow pode ser visto da seguinte forma: o utilizador envia um pergunta, que pode ser emitida por texto ou voz. Esta mensagem dá entrada na plataforma, onde o texto recebido irá ser processado. Associado ao Dialogflow existe um agente com o objetivo de converter os pedidos do utilizador em dados acionáveis. Para tal, é criado um modelo linguístico, constituído por intenções, em que cada uma contém um conjunto de frases de treino, que representam as frases exemplo que o utilizador pode dizer, ações e parâmetros, que permitem anotar entidades ou categorias de dados e as respostas de texto, visuais ou de voz criadas. Desta forma, a mensagem enviada pelo utilizador será associada a uma intenção, ou seja, a uma intenção/ação em fazer algo. Mediante essa classificação, o agente poderá enviar a resposta instantaneamente, ou então utilizar um serviço externo, com o intuito de realizar alguma operação ou de retornar um *feedback* mais coerente, através dos *fulfillments*. Um *fulfillment* é um serviço que pode ser integrado com a Dialogflow. Estes serviços, que são na maioria das vezes APIs, recebem um pedido da parte do utilizador, executando a tarefa desejada e retornando uma resposta, onde o agente é o ponto intermédio de comunicação entre estes serviços e o utilizador. Na Figura 3.8 é demonstrado um fluxo de execução, onde se enquadra esta plataforma.

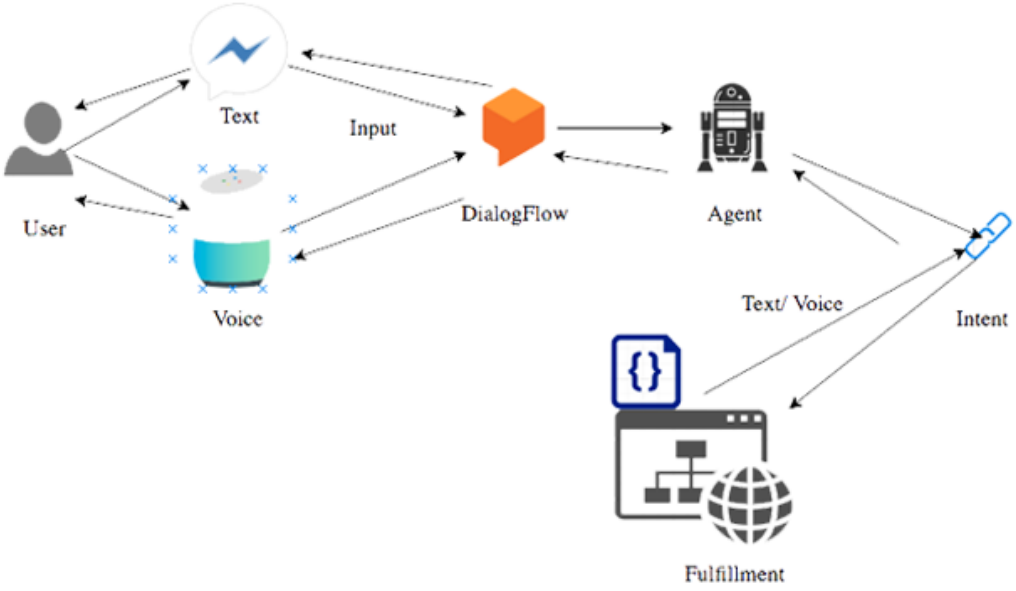

Figura 3.8: Fluxo de execução perante a utilização da Dialogflow [67]

Para o processamento e compreensão de texto, a Dialogflow suporta cerca de catorze línguas, possibilitando a utilização de dezasseis linguagens de programação distintas durante o seu desenvolvimento. Para além disso, os agentes podem ser integrados em várias plataformas de conversação, como o Slack, Facebook Messenger e Twitter, podendo também serem exportados para outros dispositivos como o Google Assistant, a Amazon Alexa e o Microsoft Cortana [68].

Rasa Stack

O Rasa Stack é um conjunto de ferramentas de ML *open-source*, que permite a criação de assistentes por voz e agentes conversacionais, capazes de realizar conversas contextuais e de responder a questões simples, feitas pelos utilizadores. De uma forma resumida, o Rasa oferece as seguintes funcionalidades [69]:

- Transformação da linguagem natural em dados estruturados;
- Aprendizagem da gestão de contextos, através de conversas reais;
- Vetores de palavras personalizados, que são treinados para um determinado domínio;
- Extração de entidades;
- Ligações entre mensagens e várias intenções;
- Aprendizagem interativa.

O Rasa Stack divide-se em dois grandes componentes: o Rasa NLU e o Rasa Core. O Rasa NLU é responsável pela compreensão da linguagem humana, onde o seu objetivo consiste em, conforme o texto analisado, prever qual a intenção associada, assim como a extração de entidades úteis através da mesma. Desta forma e mediante a questão feita pelo utilizador, o Rasa NLU dita a resposta do agente, podendo esta ser personalizada e mais completa através

da utilização das entidades extraídas. Seguindo o fluxo do *pipeline* do Rasa, o componente que se segue é o Rasa Core. Este componente é encarregue de decidir qual a ação a tomar numa conversa. Por outras palavras, é um gestor de diálogos que prevê qual a melhor ação a ser executada, com base nos dados gerados pela NLU, no histórico de conversação e nos dados de treino [70][69].

O Rasa é flexível, personalizável e facilmente intercambiável. É possível usar, tanto o Rasa Core como o Rasa NLU separadamente, onde neste último existe a possibilidade de escolher várias bibliotecas de *back-end* para o NLP. Quanto ao Rasa Core, este faz a previsão com um algoritmo baseado nas RNN, que apresenta um conjunto de resultados satisfatório. Contudo, e dada a possibilidade de personalização, esta abordagem pode ser facilmente substituída por outro algoritmo de ML [70]. Outra vantagem do Rasa é o facto de suportar todas as línguas, onde o *pipeline* pode ser usado para treinar os vetores das palavras embutidas no domínio. A acrescentar a tal, são oferecidos modelos pré-treinados em Inglês, Alemão, Espanhol, Português, Italiano, Holandês, Francês e Checo, que permitem reconhecer entidades do tipo pessoas, locais e organizações [69].

Bot School

A Bot School é uma plataforma desenvolvida pela Altice Labs, cujo objetivo consiste em ajudar na criação de assistentes virtuais, adaptados às necessidades das pequenas e médias e grandes empresas, oferecendo um canal novo, simplificado e mais natural sobre os serviços existentes [71].

Atualmente, muitas empresas lutam com a transformação digital e com a necessidade de suportar vários canais de comunicação como a *web* ou até redes sociais. Para além disso, as mesmas deparam-se com problemas em torno da IA, que é uma área que requer bastante conhecimento e esforço humano. Com base nestes incidentes, a Bot School surgiu com a necessidade de acrescentar valor, tanto aos utilizadores de assistentes virtuais, como para as empresas que os acolhem, com um tipo de implementação fácil e rápida. Resumidamente, esta plataforma foi desenhada com o simples propósito de simplificar as interações humanas no mundo digital [71]. Assim, o objetivo da Bot School é o de fornecer os seguintes serviços e funcionalidades:

- Gestão do assistente virtual;
- Facilidade no treino de diálogos;
- Integração com APIs externas;
- Suporte para as redes sociais de comunicação como o Facebook e Skype;
- Fácil integração em qualquer página *web*;
- Visão geral e análise das interações realizadas.

No que toca ao processamento da linguagem natural humana, a Bot School utiliza um NLP, também desenvolvido pela Altice Labs, chamado de Brain. Neste módulo, onde a Dialogflow

e o Rasa fazem o papel de NLU (por defeito é o Rasa), é feita a gestão de diálogos entre o agente e o utilizador, através de outro módulo chamado Context Manager API, que trata de recolher os contextos e as variáveis da conversa, associados a uma sessão. De seguida, e após a criação manual das várias intenções, entidades e frases de treino por parte do utilizador, pode ser feito o treino do modelo linguístico. Este modelo, treinado de forma supervisionada, retorna uma lista de possíveis intenções, obtidas através da NLU, com um valor de confiança associado. Desta forma, é feita a verificação dos resultados obtidos, perante os contextos encontrados, onde a intenção escolhida é utilizada para gerar uma resposta, que por sua vez pode ser dependente ou não da execução de alguma API externa ao sistema. Por fim, e após o envio da resposta, o Brain atualiza as variáveis e contextos correspondentes à última iteração. Na próxima figura é demonstrada a arquitetura da plataforma e respetivas funcionalidades.

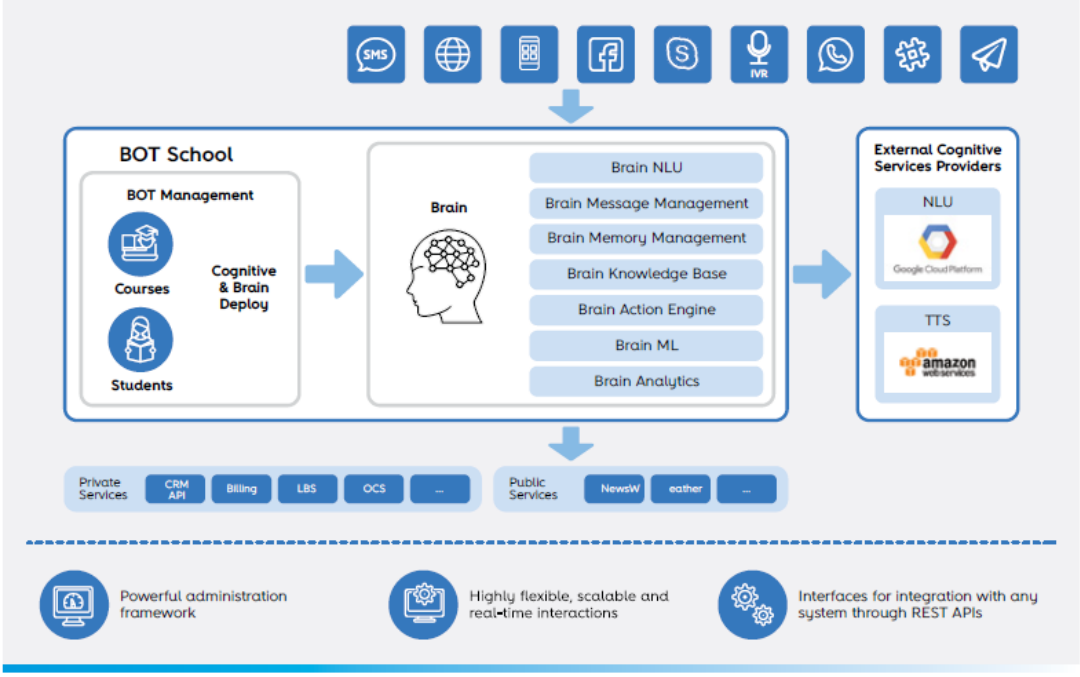

Figura 3.9: Arquitetura da Bot School [71]

Resumidamente, a Bot School é um gerador de assistentes virtuais que tem em conta três componentes importantes: cursos, estudantes e programas. Um curso é composto por um ou vários estudantes, onde cada um agrega o conteúdo proveniente de uma ou várias fontes de conhecimento independentes, chamadas de programas. A relação entre estes conceitos tem como metáfora o meio académico, que é composto por vários cursos, onde um estudante corresponde a um assistente virtual e um programa corresponde a uma disciplina. A criação, gestão e manipulação destes componentes pode ser feita através da página *web* da plataforma, onde é possível criar fluxos de execução que correspondem a uma sequência de diálogos e tarefas suportadas pelo assistente.

CAPÍTULO 4

Plataformas *Web*

O presente capítulo introduz o conceito de plataformas orientadas a componentes *web*, apresentando algumas das tecnologias mais conhecidas e utilizadas nos dias de hoje. Em primeiro lugar, é introduzido o conceito de plataformas *web*, seguido do levantamento de várias tecnologias dedicadas à construção e geração de interfaces dinâmicas e interativas. Por último, existe uma secção que menciona um conjunto de ferramentas de apoio ao projeto, de forma a garantir a persistência dos seus dados e a execução de algumas das suas funcionalidades.

4.1 Conceitos sobre Plataformas *Web*

Uma plataforma *web* é uma ferramenta de *software* que suporta a criação, desenvolvimento e publicação de aplicações e páginas *web*. Estas aplicações são criadas através de tecnologias como o HTML, CSS e JavaScript, podendo ser executadas em qualquer plataforma, desde que esta suporte um *web browser*. Com a evolução da tecnologia, nos dias de hoje é possível criar aplicações de uma forma rápida, simples e intuitiva, onde estão incluídas funcionalidades como os serviços *web*, utilização de APIs e outros recursos [72].

Atualmente, existem várias plataformas que são orientadas a componentes *web*. Estes elementos são uma coleção de padrões da *web* que permitem a criação de componentes personalizados, reutilizáveis e encapsulados. Ou seja, cada componente é um pequeno módulo independente e renovável, capaz de se relacionar com outros módulos, de forma a criar páginas e aplicações *web* completamente dinâmicas e flexíveis [73]. Por norma, estes componentes são compostos por um grupo de quatro configurações [74]:

- ***Custom Elements***: criação e definição de novos marcadores HTML, onde estes são definidos pelo autor e associados a um novo nome;
- ***Shadow DOM***: local onde os componentes são encapsulados em termos de estilo e marcações. Com este elemento, há a certeza de que nenhum estilo aplicado é substituído por outro externo;

- ***ES Modules***: definem a inclusão e a reutilização dos documentos de JavaScript de uma forma modular, baseada em padrões de alto desempenho;
- ***Templates***: definem excertos de código que não são usados durante o carregamento da página, contudo, estes podem ser instanciados mais tarde, através do respetivo ficheiro de JavaScript. Os *templates* são úteis no sentido de uma única instância de um elemento poder ser reutilizada várias vezes ao longo do projeto.

Através da utilização dos componentes *web*, é anulado o processo manual da cópia de código e de ficheiros múltiplas vezes, fazendo com que, não só seja um projeto de menor dimensão, como também é despendido menos tempo na implementação do mesmo. Com isto, é possível manter uma aplicação limpa, flexível, extensível e reutilizável, sendo relativamente simples fazer a sua manutenção e assegurar o seu correto funcionamento.

4.2 Plataformas *Web* Existentes

De forma a criar uma interação entre os resultados provenientes do último passo desta dissertação e o utilizador, torna-se necessária a criação de uma página *web*. Assim, o presente capítulo apresenta algumas plataformas que permitam construir interfaces simples e dinâmicas.

4.2.1 React

A React é uma biblioteca de JavaScript e *open-source*, utilizada para construir interfaces do utilizador. É uma biblioteca declarativa, eficiente e flexível, permitindo compor páginas complexas através da utilização de vários componentes *web* [75].

Contrariamente à grande parte das plataformas *web* que utilizam o *Model View Controller (MVC)* como um todo, a React é vista apenas como a camada *View* deste padrão. Assim, e de forma a conseguir oferecer algo mais do que um conjunto de páginas estáticas, esta biblioteca é dependente da integração de bibliotecas terceiras, como acontece no conceito da *Single Page Application (SPA)* [76]. Uma SPA é uma plataforma que carrega todo o código desenvolvido em HTML, CSS e JavaScript de uma vez só, onde a interação com o utilizador é feita numa única página, mediante a utilização dos seus componentes. Para tal, é necessário substituir a navegação por defeito dos *web browsers*, onde os URLs são geridos de forma manual, através de mecanismos chamados *routers*. Enquanto grande parte das plataformas *web* atuais tem esta funcionalidade embutida, a React necessita de integrar uma biblioteca auxiliar como o React Router [77].

Por norma, todas as aplicações *web* têm um comportamento dinâmico. Isto significa que, mediante a ocorrência de um evento, é necessário atualizar o *Domain Object Model (DOM)*, de modo a sincronizar o estado do modelo de dados com a apresentação da interface do utilizador. Esta sincronização é feita através do *two-way data binding*, ou seja, é bidirecional. No caso da React, o fluxo de dados tem um sentido unidirecional, onde a sincronização é feita com o auxílio do *Virtual DOM (V-DOM)*. Contrariamente ao DOM, que exige ao *web browser* a

execução de operações intensivas quando o seu estado é mudado, o V-DOM tem o objetivo de reduzir a quantidade de vezes que essas mesmas são feitas. Assim, sempre que o estado de um componente for alterado, a React atualiza o V-DOM, seguido da execução de um algoritmo que analisa as diferenças entre o estado atual e o anterior. Após ter sido feita esta verificação, o DOM real é atualizado, conforme as mudanças verificadas. Mediante esta abordagem, a React recolhe todos os elementos que sofreram uma alteração de estado, fazendo apenas um único pedido ao DOM, onde este faz as respetivas atualizações no mesmo instante de tempo [77].

Outra inovação desta biblioteca é a utilização do *JavaScript XML (JSX)*. O JSX é um tipo de extensão XML, ideal para lidar com os componentes. No fundo, esta tecnologia permite escrever, de uma forma declarativa, qual é o conteúdo de cada componente presente na interface do utilizador [77]. Lançada em 2013, a React é uma das ferramentas de desenvolvimento *web* mais populares, sendo usada por entidades como o Facebook, Instagram, Netflix e WhatsApp.

4.2.2 Vue.js

A Vue é uma plataforma progressiva, baseada em JavaScript, sendo utilizada na construção de interfaces do utilizador. Contrariamente às plataformas *web* semelhantes, a Vue é desenhada de raíz, de forma a que seja incrementalmente adaptável, leve e de fácil integração [78].

Tecnicamente, a Vue baseia-se na arquitetura *Model View View-Model (MVVM)*. Enquanto uma arquitetura do tipo MVC separa as *Views* dos *Models*, gerindo a sua conexão através da camada *Controller*, a MVVM permite a interação entre uma *View* e um *Model*. Esta operação é feita na camada *Model-View*, onde a *View* e o modelo de dados comunicam diretamente, através de *data bindings*. Este tipo de arquitetura é recomendada para aplicações do tipo SPA, como é o caso da Vue, onde é feita uma atualização rápida e fluída do seu conteúdo, para além da informação estar a ser continuamente guardada na Base de Dados [79].

À semelhança da React, a Vue atualiza o estado dos seus componentes com o V-DOM, contudo, o processo é feito de uma forma ligeiramente diferente. Quando é feita a modificação de um estado de um componente, a React renderiza a sub-árvore completa, onde o elemento identificado é visto como a raiz da mesma. Desta forma, são percorridos todos os seus nós-filho, através da utilização de estruturas de dados imutáveis. Quanto à Vue, as dependências de um componente, cujo estado foi alterado, são automaticamente seguidas durante a sua renderização, tal que o sistema saiba precisamente qual é o elemento que necessita de ser modificado. Mediante esta abordagem, não é requerida a utilização de quaisquer classes que ajudem na otimização da renderização, como também não existe o problema de haverem vários nós-filho a receberem atualizações de estados que não são relevantes para os mesmos [80].

Fortemente influenciada por plataformas como a AngularJs, KnockoutJS, Rivet.js e Ractive.js, a Vue oferece um conjunto valioso de alternativas, procurando encontrar o equilíbrio

entre a simplicidade e a funcionalidade [81]. Utilizando formatos como o HTML, CSS e JavaScript, há também suporte para outras tecnologias, como por exemplo, a JSX [82]. Considerada como sendo uma das plataformas *web* mais fáceis de aprender, a Vue é uma boa aliada para os diversos tipos de programadores, sendo utilizada por entidades como a Xiaomi, Gitlab, Adobe, entre outras [83].

4.2.3 Angular

A Angular é uma plataforma que facilita a construção de aplicações *web*, *mobile* e *desktop*, através da combinação de *templates* declarativos, injeção de dependências, ferramentas *end-to-end* e a integração das melhores práticas para a resolução de vários desafios de desenvolvimento. Por outras palavras, a Angular é uma tecnologia que implementa um núcleo de funcionalidades opcionais, que podem ser importadas para as aplicações criadas pelos clientes [84].

Para além das linguagens convencionais da *web*, a Angular pode ser desenvolvida em TypeScript. Esta linguagem, fortemente focada no paradigma da programação orientada a objetos e com um conjunto de tipos de dados bem definido, pode ser vista como um superconjunto do JavaScript, que possibilita o desenvolvimento de aplicações de uma forma mais limpa e simplificada. A TypeScript é uma linguagem moderna que oferece funcionalidades como o *static typing*, que faz a verificação de erros e *bugs*, antes do *script* em causa ser executado [85], a criação de classes, interfaces e módulos, assim como o *refactoring*, que contém sugestões para a melhoria da qualidade, eficiência e legibilidade do código desenvolvido pelos programadores. Adicionalmente, a TypeScript tem ainda a capacidade de ser convertida para JavaScript, de modo a que seja suportada por todos os *web browsers* [86].

Através da arquitetura da Angular, representada na Figura 4.1, verifica-se que esta é composta por quatro elementos essenciais [84]:

- **Módulos**: os módulos são um conjunto de blocos de construção básica que fornecem o contexto de compilação para os componentes, através da compilação do código relacionado nos conjuntos funcionais. Desta forma, qualquer aplicação contém pelo menos um módulo raiz, podendo ser acompanhada de outros módulos característicos;
- **Componentes**: cada aplicação Angular contém, no mínimo, um componente raiz que conecta a sua hierarquia com o DOM da página. Cada componente define uma classe que contém os dados e lógica da aplicação, sendo associado a um *template* em HTML, encarregue de gerar uma *View* a ser exibida na interface do utilizador;
- ***Templates***: um *template* combina o HTML com as marcações feitas em Angular, de forma a que seja possível fazer alterações antes destas serem exibidas. Estes elementos, para além de agruparem um conjunto de diretivas que fazem a lógica do programa, permitem a sincronização entre o modelo de dados e as interfaces do utilizador através do *two-way data binding*;
- **Serviços**: os serviços são módulos que contém dados ou lógica que não estão diretamente associados a uma *View*, mas que precisam de ser partilhados com os componentes. Para

combater esta necessidade, é feita uma injeção de dependências, permitindo que os metadados de um serviço sejam disponibilizados na classe do componente.

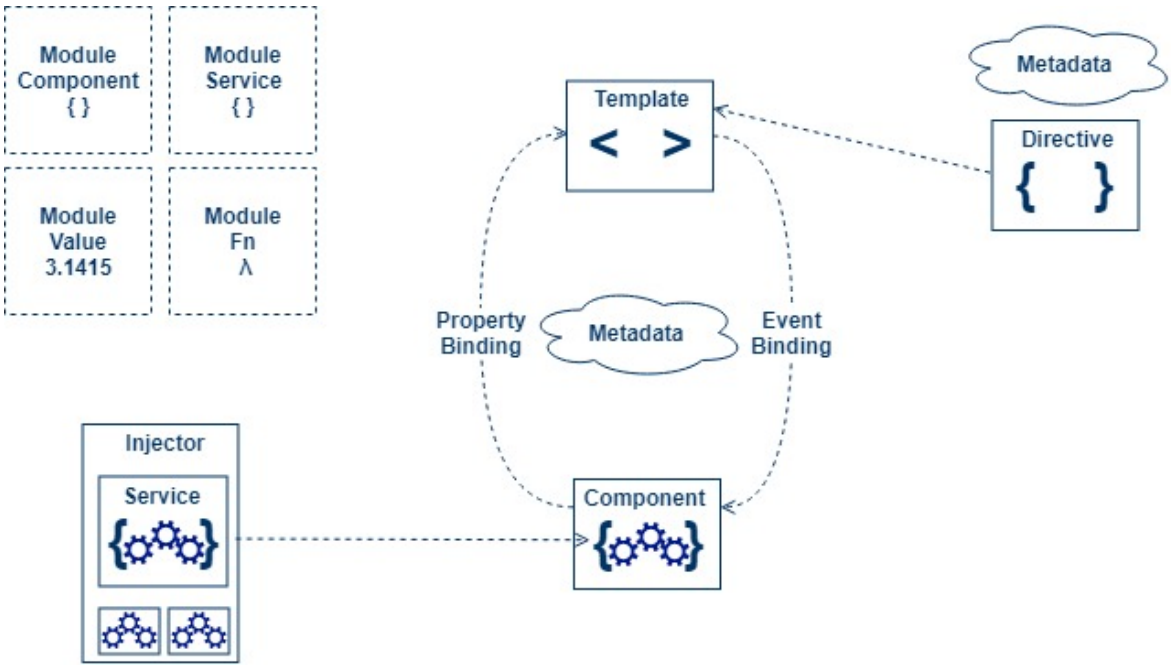

Figura 4.1: Arquitetura Angular [84]

A Angular é uma plataforma SPA, suportada por de um mecanismo de *routing* incorporado que permite fazer o carregamento e atualização dos componentes numa página só, mediante a navegação feita pelo utilizador. Desenvolvida pela Google, é atualmente uma das plataformas mais populares, onde os seus principais clientes são o Paypal, a Nike, o Freelancer e o Telegram, entre outro tipo de organizações [83].

4.2.4 WebCT

A *Web Component Toolkit (WebCT)* é uma plataforma, desenvolvida pela Altice Labs, cujo objetivo consiste em desenvolver aplicações *web* de uma forma rápida e simples, com foco na produtividade e colaboração.

Baseada na estrutura e arquitetura da Angular (Figura 4.2), esta plataforma oferece um conjunto de componentes que, cooperando entre si, conseguem automatizar o processo de desenvolvimento de páginas *web*. Este tipo de desenvolvimento é feito com o auxílio de ficheiros de configuração em JSON, chamados de *mocks*, que permitem a criação de esboços, através da definição de propriedades e atributos. Estes ficheiros são flexíveis, integráveis e podem ser reutilizados, o que faz com que não seja necessária a criação e cópia de ficheiros HTML, CSS e JavaScript. Na Figura 4.3 é apresentada uma representação de um ficheiro de configuração.

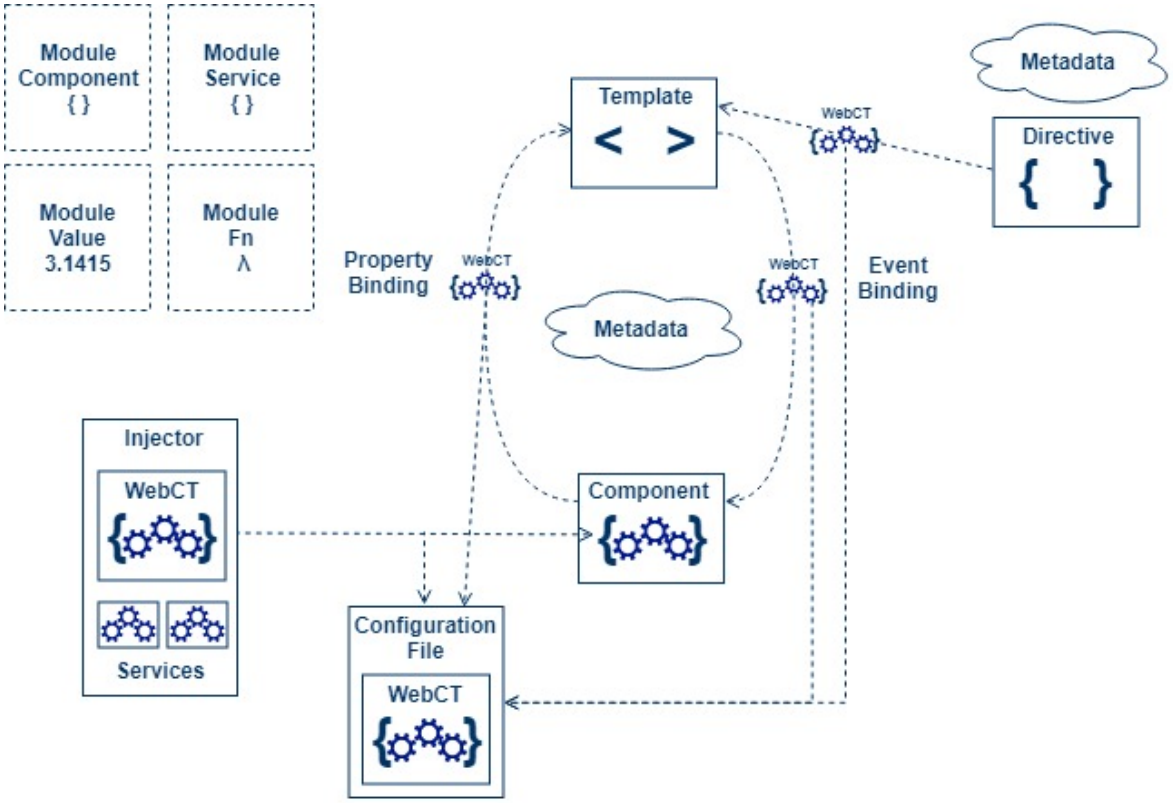

Figura 4.2: Arquitetura da WebCT

A WebCT tem um comportamento semelhante às aplicações *web* progressivas, seguindo o paradigma da nova geração de aplicações que conseguem responder aos pedidos dos mais diversos dispositivos para *web*, *mobile* e *desktop*. As aplicações *web* progressivas são um conjunto de técnicas que permitem adicionar progressivamente funcionalidades como a instanciação de componentes visuais e respetivas ligações e a integração de interfaces expostas à invocação de APIs, que antes só eram possíveis em aplicações nativas. Assim, a WebCT é composta por três componentes principais:

- **Robô**: componente responsável por dinamizar todo o processo, mais especificamente, por gerar os componentes da interface através do *parsing* feito aos ficheiros de configuração;
- **Palete**: conjunto de componentes disponíveis para incluir numa interface. Neste conjunto é possível encontrar tabelas, listas, botões, *cards*, etc;
- **Personalização**: conjuntos de componentes com características específicas que são criados quando estes são requisitados e ainda não se encontram disponíveis na palete.

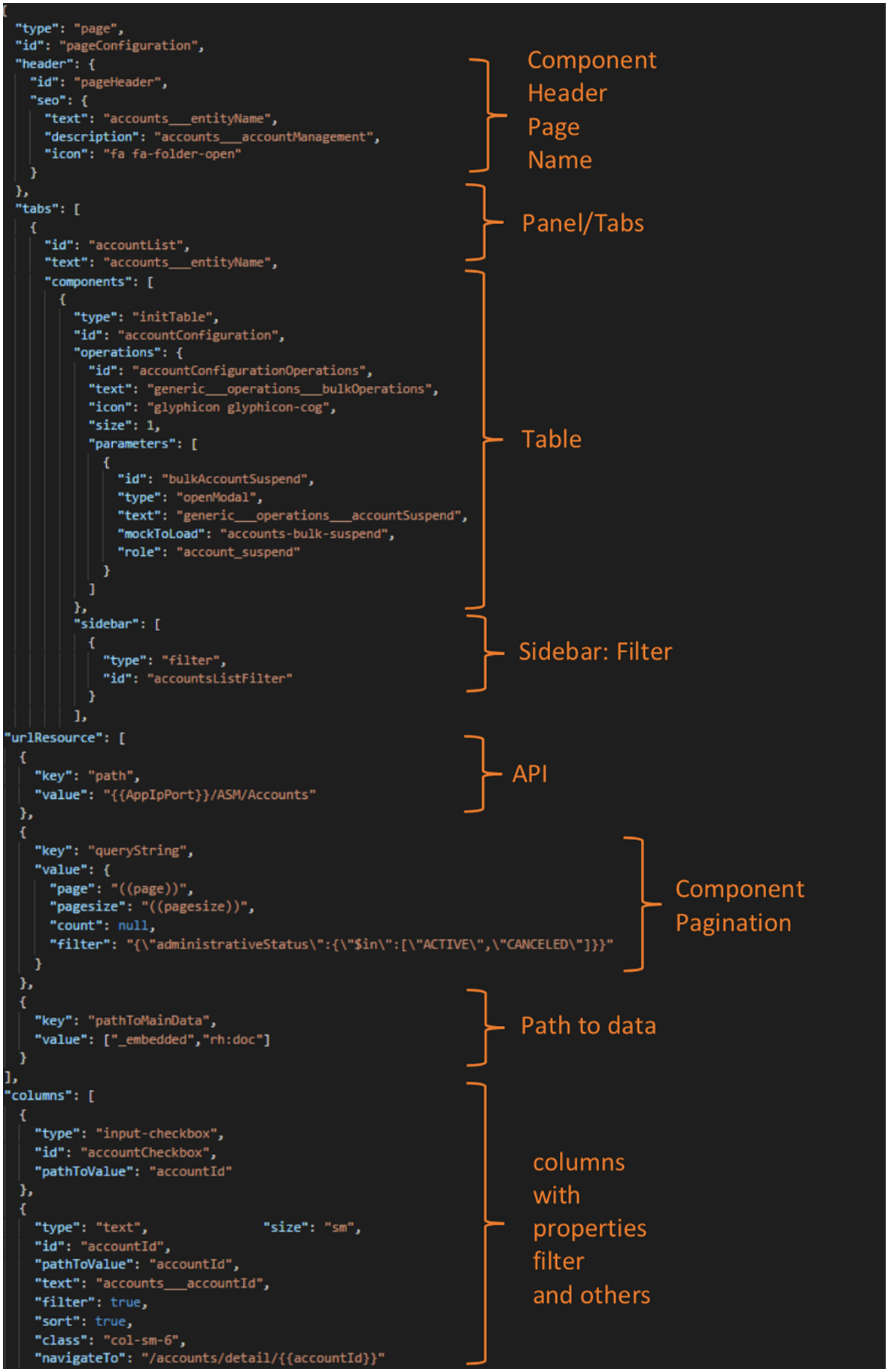

Figura 4.3: Criação de um *mock* na WebCT

Algumas vantagens da WebCT:

- acelera o processo de desenvolvimento de aplicações *web*;
- reutilização dos componentes *web*;
- são disponibilizados novos recursos e correção de *bugs* para cada aplicação *web*;
- não é necessária experiência em Angular para criar aplicações;
- os componentes são bem testados;
- *design* com uma arquitetura baseada em micro-serviços;
- fácil integração com outros sistemas através de APIs;
- "fazer menos é ganhar mais".

4.2.5 Comparativo

De forma a compreender o estado-da-arte das plataformas anteriormente apresentadas, a presente secção faz uma análise comparativa das mesmas, tendo em conta algumas propriedades.

Padrões de *Software*

A Angular é baseada na arquitetura MVC, enquanto a React é construída apenas com a camada *View*. Quanto à Vue, esta é baseada na arquitetura MVVM, focando-se na camada *View-Model*.

Gestão de Estados

A grande diferença entre a Angular e a React está na atualização dos estados dos seus componentes. Enquanto na Angular é utilizado o *two-way data binding*, atualizando simultaneamente os estados do modelo de dados e da interface do utilizador, na React é feito o *one-way data binding*, onde em primeiro lugar é feita a atualização do modelo e posteriormente são aplicadas as respetivas mudanças à interface [87]. Quanto à Vue, a mesma usufrui das duas abordagens anteriormente descritas.

Performance

Tanto a React como a Vue utilizam o modelo V-DOM, que é bastante útil em termos de *performance*. Devido à sua estrutura bem construída, a Vue oferece um melhor desempenho que o React e uma boa alocação de memória. Contudo, a Angular encontra-se num patamar superior, no que toca ao processamento de aplicações em grande escala [87].

Curva de Aprendizagem

Das três tecnologias em causa, a Vue é a que apresenta uma curva de aprendizagem menos acentuada. A capacidade em fazer um melhor aproveitamento da utilização dos componentes, a par da grande flexibilidade dada em termos de personalização, fazem com que esta plataforma seja bastante acessível. Contudo, e devido à sua simplicidade, os programadores não necessitam de seguir convenções ou quaisquer boas práticas de programação, o que poderá levar a um processo de *debugging* e/ou de testes bastante complexo [88].

Por outro lado, a plataforma que apresenta uma maior curva de aprendizagem é a Angular. Visto que esta tecnologia oferece uma solução completa, torna-se necessário dominar vários

conceitos como a arquitetura MVC e os seus componentes e a programação em JavaScript e TypeScript, sendo que esta última é uma novidade para muitos programadores [88][87].

Por último, a React apresenta uma curva de aprendizagem intermédia entre as duas tecnologias anteriores. Devido ao facto de ser uma biblioteca e não uma plataforma completa, as funcionalidades mais complexas requerem a integração de bibliotecas auxiliares. Tal situação faz com que a curva de aprendizagem não seja tão acentuada como a da Angular, contudo, a dificuldade poderá ser bem maior durante o desenvolvimento de aplicações em grande escala, onde as funcionalidades pedem inúmeras integrações de bibliotecas exteriores [88].

Concluindo, nenhuma desta tecnologias pode ser considera à priori como sendo a melhor opção. Tanto a React como a Vue e a Angular têm as suas vantagens e desvantagens, onde a sua escolha é dependente de fatores como o tipo de projeto a ser desenvolvido, quais os objetivos e requisitos a serem alcançados e qual a experiência e gostos pessoais do(s) programador(es). Por exemplo, para quem apreciar flexibilidade e preferir programar em JavaScript, deve de utilizar a React. Para quem for inexperiente e procurar a simplicidade e a separação de conceitos em projetos de pequena dimensão, Vue é a plataforma indicada. Por outro lado, para quem for fã do TypeScript e da programação orientada a objetos e/ou necessitar de desenvolver uma aplicação em grandes proporções, Angular é a escolha mais apropriada [87]. Na Tabela 4.1 é feita uma síntese das características da React, Vue e Angular.

	React	**Vue**	**Angular**
Tipo de Tecnologia	Biblioteca	Plataforma *Web*	Plataforma *Web*
Ano de Lançamento	2013	2014	2016
Criadores	Facebook	Evan You	Google
Versão Estável	16.8.6	2.6.8	7.1.4
Linguagem	JSX	HTML e JavaScript	TypeScript
Padrão de *Design*	MVC (Camada *View*)	MVVM (Camada *View-Model*)	MVC
Performance	V-DOM	V-DOM	DOM
Gestão de Estados	*One-way Data Binding*	*One-way* ou *Two-way Data Binding*	*Two-way Data Binding*
Plataforma SPA	Não	Sim	Sim
Curva de Aprendizagem	Menor que a da Angular	Pequena	Íngreme

Tabela 4.1: Tabela Comparativa entre React, Vue e Angular

4.3 Tecnologias de Suporte

A presente secção faz uma análise das possíveis tecnologias que podem auxiliar o sistema a ser desenvolvido com diversas metodologias e funcionalidades.

4.3.1 Node.js e NPM

O Node.js é um ambiente de execução *open-source* e multi-plataforma do JavaScript, onde a sua *performance* e adaptabilidade em vários tipos de projetos fazem com que esta tecnologia seja uma das mais populares [89].

Uma aplicação em Node executa num único processo, não sendo dependente do *multithreading* para executar simultaneamente vários pedidos. Desta forma, é fornecido um conjunto de primitivas I/O assíncronas da sua biblioteca por defeito, de forma a que previnam o bloqueio do código desenvolvido em JavaScript. As bibliotecas em Node são, por norma, escritas através da utilização de paradigmas não bloqueantes, o que leva a que a ocorrência de um bloqueio seja vista como uma exceção e não como uma norma. Isto permite a manipulação de várias conexões concorrentes apenas com um único servidor, sem ser necessário gerir a concorrência das *threads* [89].

O Node é leve, eficiente, escalável, tolerante a falhas e com suporte para a geração de *callbacks*, podendo ser utilizado em diversas tarefas como a construção de servidores *web*, processamento de *streams* de dados, monitorização de sistemas e acesso às Bases de Dados não-relacionais. Por outro lado, este não é adequado para lidar com sistemas de processamentos intensivos ou com servidores que trabalham com Bases de Dados relacionais [90].

O Node Package Manager (NPM) é um gestor de pacotes do Node.js, focado na publicação de projetos *open-source*. Este é constituído por módulos que consistem num conjunto de componentes públicos, reutilizáveis e disponíveis, que podem ser instalados através de um repositório *online*, com uma versão e um gestor de dependências [90].

O NPM consiste em três componentes distintos: o *website*, a interface da linha de comandos e o registo. O *website* permite a descoberta de pacotes, a configuração de perfis e a gestão de outros aspetos na experiência dos programadores, a interface da linha de comandos permite o *upload* ou *download* de pacotes e o registo é uma grande Base de Dados pública que contém todos os pacotes publicados [91].

Em suma, o NPM pode ser aplicado nos seguintes casos de uso [91]:

- Adaptar ou incorporar pacotes de código nas aplicações;
- Fazer o *download* de ferramentas independentes;
- Executar pacotes com o NPM, sem fazer nenhum *download*;
- Partilhar código com qualquer utilizador e em qualquer lugar;
- Restringir código apenas para um conjunto específico de programadores;
- Criar organizações e equipas virtuais, de forma a coordenar o desenvolvimento e a manutenção de pacotes;
- Gerir várias versões e dependências do código;
- Atualizar facilmente as aplicações;

- Encontrar outros programadores que estão de momento a trabalhar em projetos semelhantes.

4.3.2 MongoDB

A MongoDB é uma Base de Dados não-relacional (NoSQL), *open-source* e orientada a documentos, desenhada para facilitar o desenvolvimento e a sua escalabilidade.

Sendo uma Base de Dados orientada a documentos, cada registo da MongoDB é, naturalmente, um documento. Semelhantes aos objetos em JSON, estes são constituídos por um conjunto de pares chave-valor, onde os valores, para além dos tipos de dados convencionais, podem incluir outros documentos, vetores ou até listas de documentos [92]. Uma Base de Dados NoSQL é bem mais flexível do que uma relacional, pois estas não necessitam de criar qualquer tipo de relações ou restrições entre as tabelas (neste caso documentos), o que faz com que seja de fácil manutenção e edição, evitando operações custosas, como é o caso dos *joins*, que fazem interseções entre os dados de duas ou mais tabelas. Na próxima figura é apresentado um exemplo de um documento, suportado pela MongoDB.

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

Figura 4.4: Exemplo de um documento MongoDB [92]

4.3.3 RESTHeart

O RESTHeart é um servidor *open-source* de *Representational State Transfer (REST)* API para a MongoDB, onde é incorporado um servidor de HTTP de alto desempenho e sem bloqueios [93].

A criação do RESTHeart surgiu com a necessidade de o conteúdo ser armazenado e retornado de uma forma simples, uniforme e consistente, dado que grande parte da lógica de negócio existente nas aplicações SPA e móveis está a ser transferida do servidor para o lado do cliente. Sendo o JSON o formato padrão para a troca de mensagens, e com suporte na MongoDB, que é uma aplicação leve, sem esquemas complexos e orientada a documentos, a vinda do RESTHeart veio criar um mapeamento automático entre o armazenamento interno desta Base de Dados e um conjunto de recursos HTTP. Estes recursos são acessíveis externamente, onde é implementado um modelo de interação compatível com uma representação em *Hypermedia As The Engine Of Application State (HATEOAS)*, em que o estado de um processo de um cliente é inteiramente guiado por pedidos como o GET, POST, PUT, PATCH, DELETE, entre outros [93].

Resumidamente, o RESTHeart é caracterizado pelas seguintes funcionalidades [93]:

- É um servidor rápido e leve, pronto para ser usado sem qualquer código;
- É construído através de padrões como HTTP, JSON, REST e JSON *Schema*;
- Contém uma API que permite fazer *Create, Read, Update and Delete (CRUD)* aos documentos;
- Realiza operações em massa com POST, PATCH e DELETE a múltiplos documentos, através de um único pedido;
- Contém APIs para as operações do modelo de dados, de forma a criar Bases de Dados, coleções e índices através de chamadas RESTful puras;
- Faz a validação opcional dos dados com o JSON *Schema*;
- Tem um servidor estático de recursos para HTML, CSS, imagens e JavaScript e lógica aplicacional personalizada;
- É ideal para o *back-end* da React, Vue, Angular e outras *frameworks* de JavaScript.

CAPÍTULO 5

Solução Proposta e Desenvolvimento

O presente capítulo trata de descrever a solução proposta, as tecnologias utilizadas, assim como a sua arquitetura e todos os passos de implementação feitos até ao produto final. Primeiramente, é introduzida a solução proposta, acompanhada das tecnologias selecionadas para o auxílio do seu desenvolvimento. De seguida, é apresentada a arquitetura e respetivo fluxo dos dados, onde na última parte são descritos todos os blocos criados e o fluxo de execução do agente conversacional.

5.1 Solução Proposta

Tendo em conta os objetivos delineados, a solução proposta passa por conseguir criar um sistema que permita fazer a conversão de áudio para texto, seguida da análise semântica dos seus diálogos. É durante o processo da análise do texto que são feitas duas classificações: a primeira tenta classificar e criar associações entre frases e intenções/ações, enquanto a segunda é dedicada ao reconhecimento de entidades. Por último, é requerida a construção de interfaces do utilizador, através de plataformas *web* orientadas a componentes, de modo a que o sistema seja capaz de recomendar frases de treino, entidades e variáveis a acrescentar dinamicamente à base de conhecimento do agente conversacional.

5.2 Tecnologias Selecionadas

Para implementar a solução proposta, foram escolhidas várias tecnologias e métodos eficientes, flexíveis, modulares e gratuitos que auxiliaram o seu desenvolvimento, de forma a cumprir os objetivos traçados.

5.2.1 Bot School

A Bot School foi criada com o objetivo de facilitar o processo da criação de assistentes virtuais. Dada a sua facilidade de utilização, na qual é possível desenvolver fluxos de execução para

os assistentes e analisar os mesmos, esta plataforma foi utilizada com esse mesmo propósito, tendo também em conta o suporte para receber sugestões de novas frases e entidades.

5.2.2 pyAudioAnalysis e Pydub

Das tecnologias de análise de áudio encontradas, poucas oferecem as suas funcionalidades gratuitamente. Dentro desse lote restrito, a pyAudioAnalysis foi a biblioteca escolhida, pelo facto de esta ser simples e adaptável e capaz de fazer a segmentação de áudio com a *Speaker Diarization (SD)*, entre dois ou mais oradores, de uma forma não supervisionada.

Quanto à Pydub, esta possibilita a manipulação do áudio, realizando várias operações como alterar o volume e a velocidade de reprodução de um ficheiro de som, onde é também possível dividir o mesmo, através de um intervalo de tempo especificado. Esta última funcionalidade é útil pois, mediante os resultados provenientes da SD, é possível dividir o registo original em vários ficheiros de menor duração.

5.2.3 Rasa NLU e Identificador de Entidades

Para a compreensão da linguagem natural, a tecnologia escolhida foi a Rasa NLU, pela razão desta ser uma ferramenta flexível e com suporte para a personalização dos seus modelos. Para além disso, a Rasa contém mecanismos de classificação, através de algoritmos de ML, permitindo associar o texto analisado a um grupo de intenções/ações.

De todas as tecnologias de análise de texto estudadas, as que suportam a identificação de entidades na língua portuguesa só conseguem reconhecer categorias genéricas como pessoas, organizações e locais, sendo bastante limitadas por não conseguirem identificar, por exemplo, distritos e concelhos de Portugal. Em consequência disso e mediante as necessidade de trabalhar com termos portugueses bastante característicos, a par de alguns estrangeirismos, foi criado um identificador de entidades próprio, que contém um conjunto de funcionalidades, baseadas em ferramentas e métodos de NLP.

5.2.4 WebCT e Angular

A WebCT foi desenvolvida a pensar numa arquitetura baseada em micro-serviços. Isto permite a criação de pequenos componentes modulares e reutilizáveis e a integração facilitada de APIs, desenvolvidas externamente. Por estas razões, a WebCT e consequentemente a Angular foram as plataformas escolhidas para criar interfaces *web* dinâmicas e interativas.

5.2.5 Node.js e NPM

O Node.js foi o ambiente de execução escolhido. Este, para além da sua *performance* bastante satisfatória e da sua adaptabilidade a vários tipos de projetos, é baseado num modelo assíncrono, de forma a evitar o bloqueio dos processos. Ideal para tarefas como o acesso e manipulação das Bases de Dados não-relacionais, foi também tido em conta o facto de o mesmo ser bastante utilizado na Angular.

Quanto ao NPM, este foi o gestor de pacotes eleito, visto que é instalado automaticamente com o Node.js.

5.2.6 MongoDB

Dada a sua flexibilidade e criação de documentos, semelhantes a objetos JSON, a MongoDB foi a Base de Dados escolhida para guardar os dados utilizados. A utilização desta Base de Dados orientada a documentos permite fazer uma maior abstração das dependências e restrições entre as coleções existentes, resultando numa manipulação e edição simplificada do seu conteúdo.

5.2.7 RESTHeart

No contexto deste projeto, o RESTHeart é o *middlware* utilizado para fazer a exposição e a comunicação dos dados no sistema.

5.3 Arquitetura

Na Figura 5.1 é apresentada a arquitetura da solução proposta, composta por quatro blocos principais: o agente conversacional, o *Speech-to-Text (STT)*, o identificador de entidades e o criador de sugestões. No primeiro bloco, é definido um agente conversacional, auxiliado por uma base de conhecimento que permite ao mesmo compreender as intenções do cliente, gerando respostas através da linguagem natural humana. Quanto ao bloco STT, este é encarregue de converter todo o áudio analisado em texto, tendo suporte para a transcrição de ficheiros. No bloco identificador de entidades, o objetivo, mediante o texto recebido, consiste em identificar os termos mais relevantes e associá-los a uma categoria pré-definida. Por último, o criador de sugestões é encarregue de sugerir conteúdo que possa ser adicionado dinamicamente à base de conhecimento do agente. Existe ainda o módulo independente da classificação de intenções, que tem a responsabilidade de encontrar correspondências entre as respostas analisadas e as intenções/ações existentes na base de conhecimento.

Em termos de implementação, os blocos e módulos apresentados foram definidos da seguinte forma:

- O agente conversacional foi criado através da plataforma Bot School;
- O STT auxilia-se da pyAudioAnalysis e da pyDub, tendo sido desenvolvida a sua lógica em Python;
- O identificador de entidades encontra-se implementado em Java, sendo um projeto do tipo Maven;
- O criador de sugestões foi desenvolvido em Node.js;
- O classificador de intenções, apesar de ter sido desenvolvido em Node.js, auxilia-se das funcionalidades disponíveis na Rasa NLU.

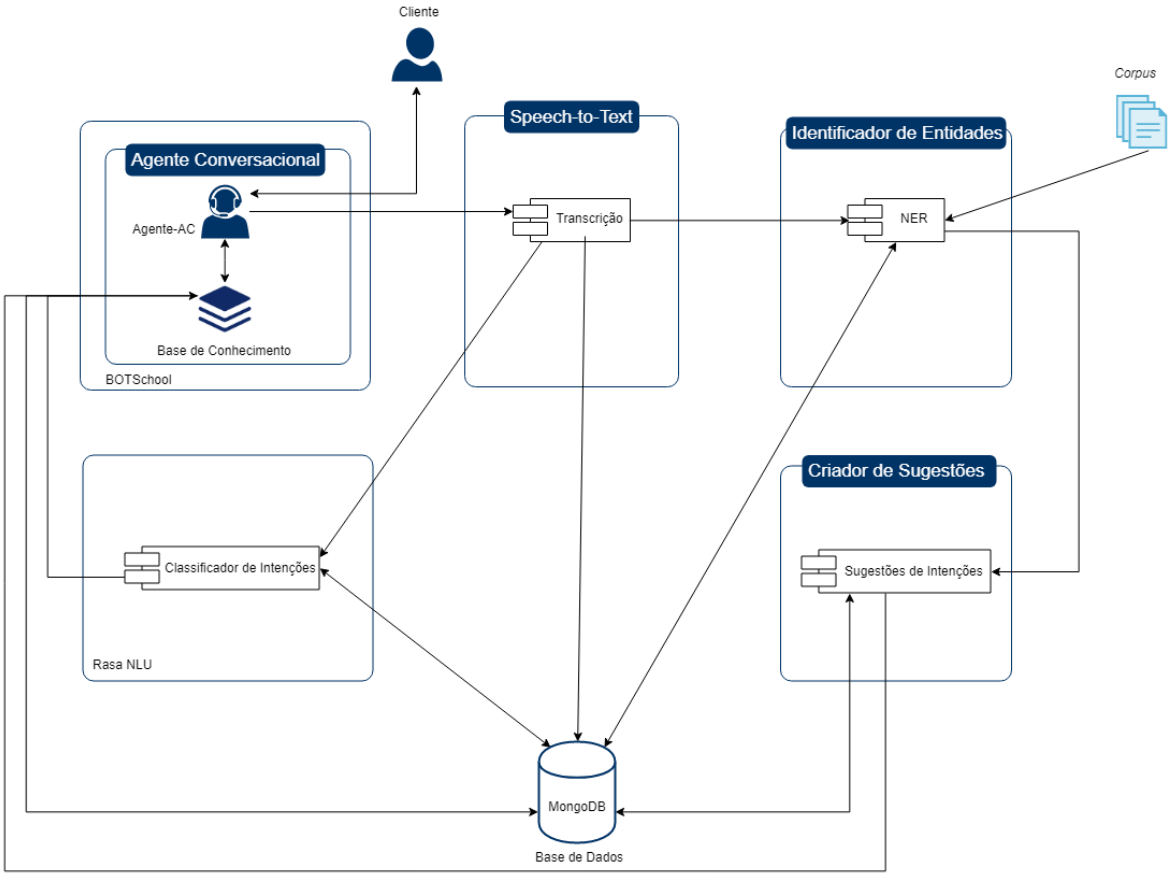

Figura 5.1: Arquitetura da Solução Proposta

O fluxo de dados da solução proposta é compreendido pelas seguintes etapas:

1. **Definição do agente conversacional e da sua base de conhecimento**
   Na solução proposta é definido um agente conversacional, cujo objetivo é o de conseguir comunicar com o cliente e realizar várias tarefas, conforme os pedidos recebidos. Toda a inteligência e capacidade de diálogo do agente é definida numa base de conhecimento especializada. É nesta fonte que se encontra toda a lógica e conhecimento do domínio do problema, onde são configuradas várias respostas, construídas de forma personalizada para cada situação possível;

2. **Conversão de áudio para texto**
   A fonte de comunicação principal do agente é o texto. Assim, e derivado da possibilidade do cliente poder comunicar por voz, o áudio analisado é convertido para texto, através do conceito STT. Esta funcionalidade oferece a oportunidade dos outros blocos da solução fazerem uma correspondência entre a mensagem recebida e a base de conhecimento definida no agente. Neste bloco, é também suportada a transcrição de registos de áudio, aplicando o conceito da SD, que permite dividir os diálogos feitos entre o operador e o cliente.

3. **Classificação de intenções**
   Mediante a resposta recebida da parte do cliente, é tentada a criação de uma correspondência entre a frase analisada e o conjunto de intenções, definido na base de conhecimento do agente. No caso desta correspondência ser bem sucedida, é despoletada uma resposta e/ou ação, configurada especificamente para a situação detetada, caso contrário, o agente devolve uma incompreensão, que dá-se pelo nome de *fallback*. Todo o histórico de diálogos feitos entre o agente e o cliente é armazenado na Base de Dados, de forma a serem analisados nas etapas subsequentes.

4. **Identificação de entidades**
   Do histórico de diálogos feitos com o cliente, são analisadas as interações cujas frases não foram compreendidas pelo agente. Deste conjunto, cada instância é enviada para o identificador de entidades, onde o seu objetivo passa por analisar novamente o texto, tentando procurar por palavras-chave que estejam associadas a alguma categoria pré-definida.

5. **Criação de sugestões**
   A última etapa passa por criar sugestões que permitam adicionar dinamicamente conteúdo à base de conhecimento suportada pelo agente. Os resultados anteriormente obtidos no identificador de entidades são confrontados com os dados definidos na base de conhecimento. No caso do seu conteúdo ser semelhante, é sugerido que a frase em questão seja adicionada dinamicamente ao conjunto de dados em causa. De momento, a confirmação deste tipo de sugestões é feita, de forma supervisionada, pelo administrador do agente.

5.4 Modelo de Dados

Durante o fluxo de execução do agente, a informação é constantemente modificada. Como sequência da realização de várias tarefas, vários conjuntos de dados são criados, editados, partilhados e removidos a qualquer momento, podendo estes ser enviados e/ou recebidos através das mais diversas fontes. De forma a atender a este pedido, e pelo facto de alguns métodos estarem dependentes dos vários parâmetros de entrada, todos os dados gerados são armazenados numa Base de Dados MongoDB, sendo a sua conexão feita através de pedidos HTTP.

Com o objetivo de armazenar dados, foram criadas e utilizadas as seguintes coleções: "Students", "Programs", "DialogueHistory", "Dialogues", "Entities", "Stopwords" e "Intents-Suggestions". Na próxima figura é apresentado o Diagrama de Classes da solução proposta, composto por estas coleções, sendo os seus atributos detalhados nas secções seguintes.

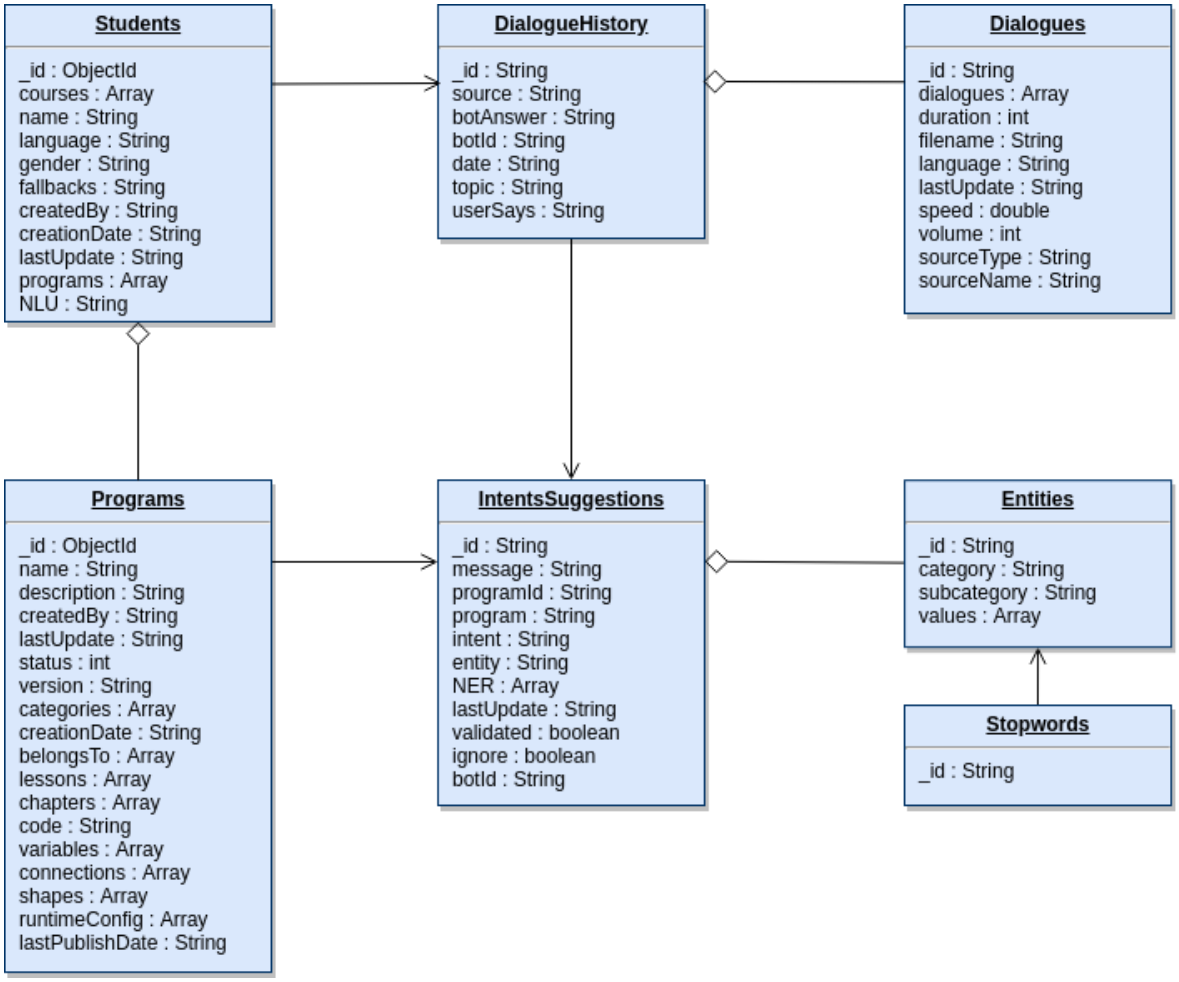

Figura 5.2: Diagrama de Classes da Solução Proposta

5.4.1 Coleção "Students"

A coleção "Students" guarda toda a informação referente ao agente conversacional. Na próxima tabela são apresentados alguns dos atributos que compõem este conjunto de dados.

Atributo	Tipo	Descrição
_id	ObjectId	Identificador do agente
courses	Array	Cursos aos quais o agente pertence
name	String	Nome do agente
language	String	Língua compreendida pelo agente
gender	String	Sexo do agente
fallbacks	Array	Configuração de *fallbacks* do agente
createdBy	String	Nome de utilizador do seu criador
creationDate	String	Data de criação
lastUpdate	String	Última data de atualização
programs	Array	Base de conhecimento associada
NLU	String	Tipo de NLU utilizada

Tabela 5.1: Coleção de dados relativa ao agente conversacional

5.4.2 Coleção "Programs"

A coleção "Programs" guarda toda a informação da base de conhecimento do agente conversacional. Na próxima tabela são apresentados alguns dos atributos que compõem este conjunto de dados.

Atributo	Tipo	Descrição
_id	ObjectId	Identificador da base de conhecimento
name	String	Nome da base de conhecimento
description	String	Descrição da base de conhecimento
createdBy	String	Nome de utilizador do seu criador
lastUpdate	String	Última atualização da base de conhecimento
status	Int	Estado atual da base de conhecimento
version	String	Versão atual da base de conhecimento
categories	Array	Categorias onde a base de conhecimento se enquadra
creationDate	String	Data de criação da base de conhecimento
belongsTo	Array	Agentes que possuem a base de conhecimento
lessons	Array	Intenções e frases de treino aprendidas na base de conhecimento
chapters	Array	Entidades e sinónimos criados na base de conhecimento
code	String	Código identificativo da base de conhecimento
variables	Array	Variáveis criadas na base de conhecimento
connections	Array	Conexões criadas no fluxo de execução
shapes	Array	Objetos utilizados no fluxo de execução
runtimeConfig	Array	Configurações de cada intenção na base de conhecimento
lastPublishDate	String	Última data de publicação da base de conhecimento

Tabela 5.2: Coleção de dados relativa à base de conhecimento

5.4.3 Coleção "DialogueHistory"

A coleção "DialogueHistory" guarda todo o histórico de conversação feita entre o cliente e o agente conversacional. Na próxima tabela são apresentados alguns atributos que compõem este conjunto de dados.

Atributo	Tipo	Descrição
_id	Object	Identificador do diálogo
source	String	Fonte de dados do diálogo
botAnswer	String	Resposta dada pelo agente
botId	String	Identificador do agente
date	String	Data da interação
topic	String	Intenção da base de conhecimento identificada
userSays	String	Mensagem do utilizador

Tabela 5.3: Coleção de dados relativa ao histórico de diálogos

5.4.4 Coleção "Dialogues"

A coleção "Dialogues" guarda toda a informação referente às transcrição de áudio feitas pelo módulo STT. Neste conjunto, o atributo *dialogues* é compreendido pelos seguintes campos:

- ***index***: índice do diálogo;
- **speaker**: número do orador (0 - Operador e 1 - Cliente);
- ***initialTime***: tempo de reprodução (segundos) já realizado quando é iniciado o diálogo;
- ***finalTime***: tempo de reprodução (segundos) já realizado quando é finalizado o diálogo;
- ***text***: texto transcrito do diálogo.

Na próxima tabela é apresentado o conjunto de atributos que compõem esta coleção.

Atributo	Tipo	Descrição
_id	String	Identificador da transcrição
dialogues	Array	Conjunto de diálogos transcritos
duration	Int	Duração do ficheiro de áudio (segundos)
filename	String	Pasta onde se encontra o ficheiro transcrito
language	String	Língua compreendida
lastUpdate	String	Data da última atualização
speed	Double	Velocidade de reprodução do ficheiro
volume	Int	Mudança de volume aplicada ao áudio
sourceType	String	Tipo de agente utilizado
sourceName	String	Nome do agente

Tabela 5.4: Coleção de dados relativa aos diálogos transcritos

5.4.5 Coleção "Entities"

A coleção "Entities" guarda toda a informação referente às categorias de entidades pré-definidas, assim como os seus valores. Na próxima tabela é apresentada a composição dos atributos deste conjunto de dados.

Atributo	Tipo	Descrição
_id	String	Identificador da entidade
category	String	Categoria à qual a entidade pertence
subcategory	String	Subcategoria à qual a entidade pertence
values	Array	Sinónimos da entidade

Tabela 5.5: Coleção de dados relativa às entidades

5.4.6 Coleção "Stopwords"

Durante o processamento de texto, existem alguns termos que são bastante frequentes e irrelevantes para a classificação das entidades. Por norma, esta regra aplica-se a determinantes, pronomes e artigos definidos e indefinidos, onde estes compõem uma lista de palavras a excluir do texto, chamadas de *stopwords*. Assim, a coleção "Stopwords" tem o objetivo de guardar todas as palavras irrelevantes, de forma a que possam ser posteriormente removidas das frases analisadas. Na próxima tabela é apresentado o conteúdo que compõe este conjunto de dados.

Atributo	Tipo	Descrição
_id	String	Valor da *stopword*

Tabela 5.6: Coleção de dados relativa às *stopwords*

5.4.7 Coleção "IntentsSuggestions"

A coleção "IntentsSuggestions" guarda toda a informação, referente às sugestões criadas. Na próxima tabela é apresentado o conteúdo que compõe este conjunto de dados.

Atributo	Tipo	Descrição
_id	String	Identificação da sugestão
message	String	Frase de treino proposta
programId	String	Identificador da base de conhecimento para a qual é feita a sugestão
program	String	Nome da base de conhecimento para a qual é feita a sugestão
intent	String	Nome da intenção na base de conhecimento para a qual é feita a sugestão
entity	String	Nome da categoria de entidades encontrada
NER	Array	Conjunto de entidades encontradas no identificador de entidades
lastUpdate	String	Data da última atualização
validated	Boolean	Sugestão validada pelo administrador
ignore	Boolean	Sugestão ignorada pelo administrador
botId	String	Identificador do agente para o qual é feita a sugestão

Tabela 5.7: Coleção de dados relativa às sugestões

5.5 Agente Conversacional e Base de Conhecimento

O "Agente-AC" foi criado com o intuito de poder comunicar com o cliente, através da linguagem natural humana, de forma a realizar os possíveis pedidos, feitos pelo mesmo. Por outras palavras, o "Agente-AC" foi desenvolvido com o objetivo de prestar apoio ao cliente nas mais diversas áreas e aplicações, conforme a definição do domínio do problema.

Toda a lógica envolvente no comportamento do agente, desde a compreensão do problema analisado, a composição do conjunto de respostas/ações a serem feitas e a gestão do seu fluxo conversacional é feita numa base de conhecimento. Associado ao "Agente-AC", encontra-se a base de conhecimento "ApoioCliente" que, como o próprio nome sugere, tem como objetivo prestar apoio ao cliente em alguns problemas recorrentes na área das telecomunicações, mais especificamente, assuntos relacionados com saldo, obtenção do *pin* e do *puk* de um cartão e a solicitação de uma segunda via de uma fatura.

Quanto ao seu conteúdo, a base de conhecimento "ApoioCliente" é composta pelas seguintes intenções e frases de treino:

- "**Saudações**": intenção que pretende saudar, dando as boas-vindas;
- "**Saber Saldo**": intenção que faz o pedido de verificação do saldo;
- "**Dizer Número**": intenção que corresponde ao fornecimento do número do telemóvel;

- "**Saber Pin e Puk**": intenção que faz o pedido de fornecimento do *pin* e do *puk*;
- "**Segunda Via de Fatura**": intenção que corresponde ao pedido da segunda via da fatura;
- "**Dizer Morada**": intenção que corresponde ao fornecimento da morada de residência;
- "**Agradecimentos**": intenção que trata de agradecer a algo ou alguém;
- "**Confirmação**": intenção que tem o objetivo comprovar e/ou certificar algo;
- "**Negação**": intenção que tem o objetivo não comprovar ou desacordar sobre algo.

Na Figura 5.3 são apresentadas algumas frases de treino, associadas às intenções definidas. Para além disso, a "ApoioCliente" contém ainda uma variável, referente à morada de residência inserida pelo cliente e a entidade "**pin-puk**", que é composta pelos sinónimos "**pin**", "**puk**" e "**pin e puk**", sendo utilizada na intenção "**Saber Pin e Puk**".

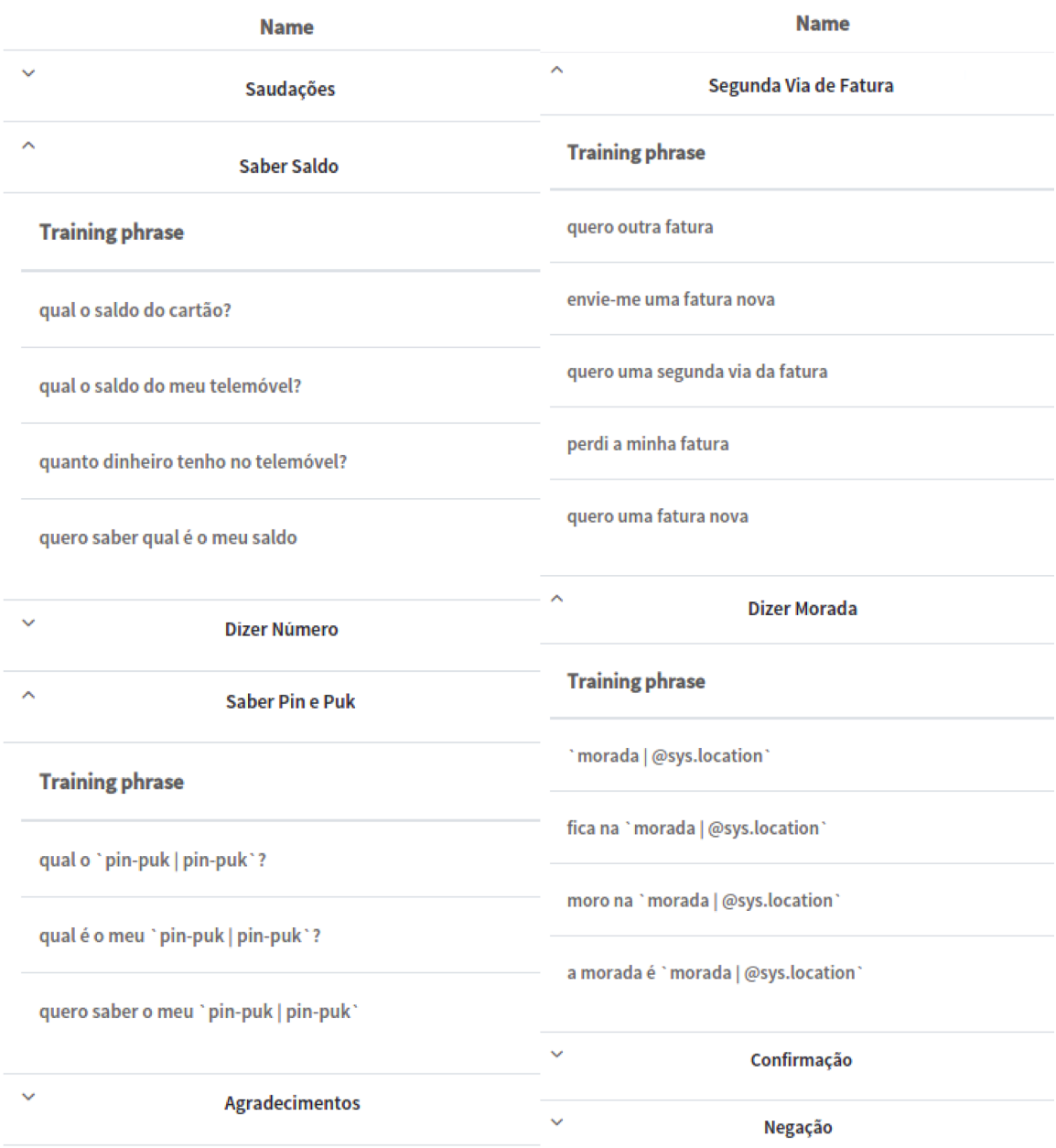

Figura 5.3: Intenções e frases de treino definidas na base de conhecimento

5.6 Fluxo Conversacional do Agente

Na Figura 5.4 é apresentado o fluxo conversacional do agente. Este tipo de fluxo é visto como um grafo orientado, composto por vários nós que correspondem às respostas a serem enviadas pelo agente e por um conjunto de arestas que representam as intenções definidas e posteriormente classificadas.

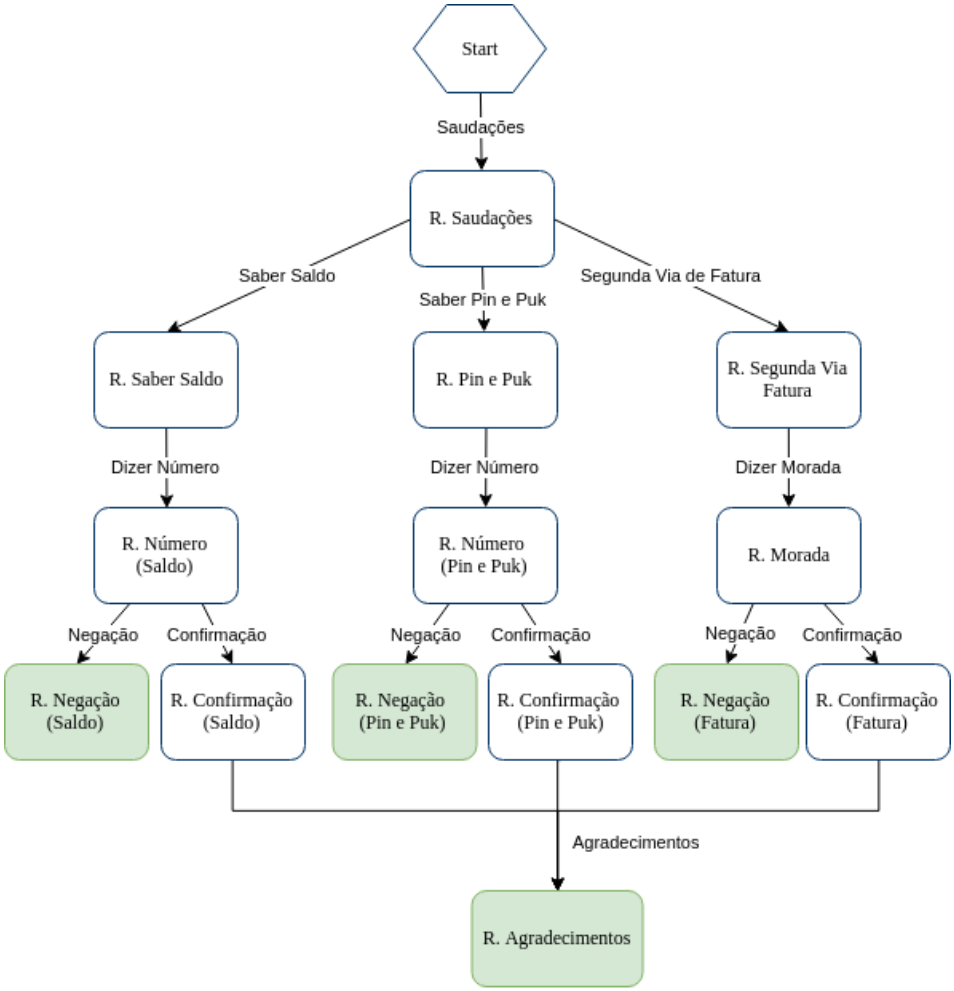

Figura 5.4: Fluxo conversacional do agente

Com exceção do nó raiz, todos os nós existentes neste grafo orientado são dependentes do seu nó-pai. Em consequência desta restrição e com o objetivo de enviar a resposta apropriada, o agente tem que completar um caminho, composto por um conjunto de nós, que foi especificamente configurado para representar um determinado caso de uso. Assim, o fluxo conversacional do agente tem como raiz o nó "**Start**", enquanto os nós-folha são os nós a verde, encarregues de finalizar a conversa com o cliente. Quanto aos casos de uso, são suportadas quatro situações distintas: saudações, assuntos relacionados com o saldo, obtenção do *pin* e *puk* de um cartão e solicitação da segunda via da fatura.

5.6.1 Saudações

As saudações/boas-vindas trocadas entre o agente e o cliente são o primeiro passo na conversação entre ambos. Por norma, o cliente cumprimenta o agente e o mesmo dá-lhe as boas-vindas, dando origem a um dos casos de uso seguintes. A transição do nó raiz para este nó é representada pela intenção "**Saudações**".

5.6.2 Assuntos Relacionados com o Saldo

Nos assuntos relacionados com o saldo, o cliente pergunta ao agente sobre o saldo atual do seu cartão. Após terem sido feitas as saudações, este caso de uso realiza os seguintes passos:

1. O cliente questiona o agente sobre qual o saldo atual do seu cartão (intenção "**Saber Saldo**");
2. O agente pergunta ao cliente qual é o número de telemóvel para o qual pretende saber o saldo (resposta "**R. Saber Saldo**");
3. O cliente diz qual é o número de telemóvel para o qual pretende saber o saldo atual (intenção "**Dizer Número**");
4. O agente repete o número compreendido e pede a confirmação do cliente (resposta "**R. Número (Saldo**)");
5. O cliente confirma, ou não, o número compreendido pelo agente. No caso do número ser confirmado (intenção "**Confirmação**"), o agente responde qual o saldo atual do cartão do cliente (resposta "**R. Confirmação (Saldo)**"). Caso contrário (intenção "**Negação**"), o agente encaminha a chamada para um assistente humano (resposta "**R. Negação (Saldo)**");
6. No final do agente indicar o saldo atual do cartão, o cliente pode agradecer o serviço prestado (intenção "**Agradecimentos**"), dando por terminado o caso de uso.

Na próxima figura é apresentado um exemplo de uma conversação feita entre o agente e o cliente, de forma a que o último saiba qual é o saldo do seu cartão.

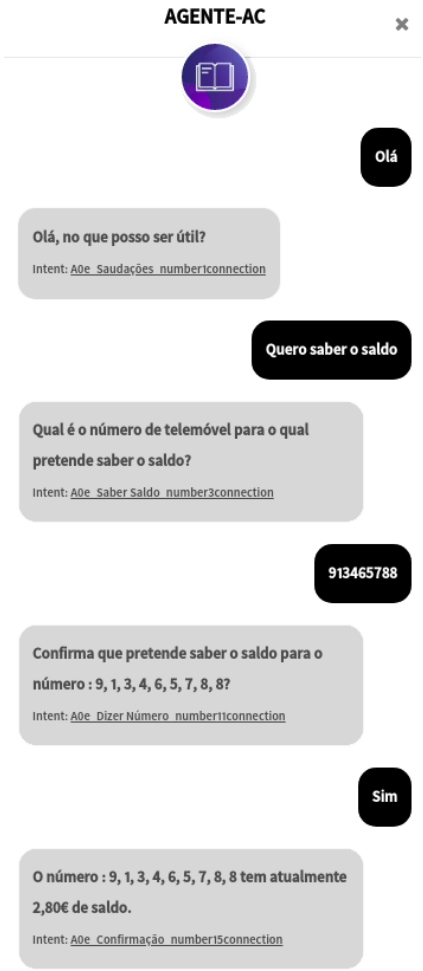

Figura 5.5: Conversação feita entre o agente e o cliente sobre o saldo no cartão

5.6.3 Obtenção do *Pin* e *Puk* de um Cartão

No caso de uso referente à obtenção do *pin* e *puk*, o cliente pretende desbloquear o cartão que se encontra a ser utilizado no seu telemóvel. Após terem sido feitas as saudações, este caso de uso realiza os seguintes passos:

1. O cliente pergunta qual o *pin* e/ou *puk* do seu cartão (intenção "**Saber Pin e Puk**");
2. O agente pergunta ao cliente qual é o número de telemóvel para o qual pretende saber o *pin* e/ou *puk* (resposta "**R. Pin e Puk**");
3. O cliente diz qual é o número de telemóvel para o qual pretende saber o *pin* e/ou *puk* (intenção "**Dizer Número**");
4. O agente repete o número compreendido e pede a confirmação do cliente (resposta "**R. Número (Pin e Puk**)");
5. O cliente confirma, ou não, o número compreendido pelo agente. No caso do número ser confirmado (intenção "**Confirmação**"), o agente responde qual o *pin* e o *puk* do cartão (resposta "**R. Confirmação (Pin e Puk)**"). Caso contrário (intenção "**Negação**"), o agente encaminha a chamada para um assistente humano (resposta "**R. Negação (Pin e Puk)**");

6. No final do agente indicar o *pin* e o *puk* do cartão, o cliente pode agradecer o serviço prestado (intenção "**Agradecimentos**"), dando por terminado o caso de uso.

5.6.4 Solicitação da Segunda Via da Fatura

Na solicitação da segunda via da fatura, o cliente pede ao agente uma segunda cópia de uma determinada fatura. Após terem sido feitas as saudações, este caso de uso realiza os seguintes passos:

1. O cliente pede uma segunda via da fatura (intenção "**Segunda Via de Fatura**");
2. O agente pede a morada ao cliente (resposta "**R. Morada**");
3. O cliente diz qual é a sua morada de residência (intenção "**Dizer Morada**");
4. O agente repete a morada compreendida e pede a confirmação do cliente (resposta "**R. Morada**");
5. O cliente confirma, ou não, a morada compreendida pelo agente. No caso da morada ser confirmada (intenção "**Confirmação**"), o agente emite uma fatura nova (resposta "**R. Confirmação (Fatura)**"). Caso contrário (intenção "**Negação**"), o agente encaminha a chamada para um assistente humano (resposta "**R. Negação (Fatura)**");
6. No final do agente emitir a segunda via da fatura, o cliente pode agradecer o serviço prestado (intenção "**Agradecimentos**"), dando por terminado o caso de uso.

5.6.5 Frases de Treino Compreendidas pelo Agente Conversacional

Os casos de uso suportados pelo agente são aqueles cujas intenções são definidas na sua base de conhecimento e planeadas no seu fluxo conversacional. Para tal, cada intenção é composta por um conjunto de frases de treino, o que permite ao agente compreender os pedidos recebidos pelo cliente, respondendo-lhe de forma adequada. Eis alguns exemplos de frases de treino suportadas:

- Intenção "**Saudações**"
  - "Olá";
  - "Viva, tudo bem?";
  - "Oi".
- Intenção "**Saber Saldo**"
  - "Qual o saldo do meu telemóvel?";
  - "Qual o saldo do cartão?";
  - "Quero saber qual é o meu saldo".
- Intenção "**Saber Pin e Puk**"
  - "Quero saber o meu pin";
  - "Qual o pin?";
  - "Qual é o meu puk?".
- Intenção "**Segunda Via de Fatura**"
  - "Quero uma segunda via da fatura";
  - "Envie-me uma nova fatura";
  - "Perdi a minha fatura".

- Intenção ““**Dizer Número**”
  - “913465788”;
  - “É o número 913465788”.
- Intenção “**Dizer Morada**”
  - “Rua XPTO”;
  - “Moro na Rua XPTO”.
- Intenção “**Confirmação**”
  - “Sim”;
  - “Confirmo”;
  - “Concordo”.
- Intenção “**Negação**”
  - “Não”;
  - “Não confirmo”;
  - “Discordo”.
- Intenção “**Agradecimentos**”
  - “Muito obrigado pela ajuda prestada”;
  - “Obrigado”.

5.6.6 Respostas Enviadas pelo Agente Conversacional

Na base de conhecimento do agente conversacional estão configuradas as seguintes respostas:

- “**R. Saudações**”: “Olá, no que posso ser útil?”;
- “**R. Saber Saldo**”: “Qual é o número de telemóvel para o qual pretende saber o saldo?”;
- “**R. Pin e Puk**”: “Qual é o número para o qual pretende saber o pin e o puk?”;
- “**R. Segunda Via Fatura**”: “Diga-me por favor a morada da sua residência.”;
- “**R. Número (Saldo)**”: “Confirma que pretende saber o saldo para o número 913465788?”;
- “**R. Número (Pin e Puk)**”: “Confirma que pretende saber o pin e o puk para o número 913465788?”;
- “**R. Confirmação (Saldo)**”: “O número 913465788 tem atualmente 2,80€ de saldo.”;
- “**R. Confirmação (Pin e Puk)**”: “O pin para o número 913465788 é o 3428 e o puk é o 013821.”;
- “**R. Confirmação (Fatura)**”: “Enviaremos a segunda via da fatura o mais breve possível.”;
- “**R. Negação (Saldo)**”, “**R. Negação (Pin e Puk)**” e “**R. Negação (Fatura)**”: “Dada a inconformidade do pedido, vou passar a chamada para um assistente. Obrigado.”;
- “**R. Agradecimentos**”: “Sempre ao seu dispor. Obrigado.”.

5.7 *Speech-to-Text*

O bloco STT é responsável por compreender o que é dito pelo cliente, podendo também fazer a transcrição de um ficheiro de áudio para texto, após a sua requisição. Recebido o pedido, o primeiro passo consiste em verificar se o ficheiro indicado é válido. Para os ficheiros deste tipo serem lidos, os mesmos têm que estar armazenados na pasta por defeito, a *data/audio_files*, que se encontra no sistema de ficheiros do presente bloco. Desta forma, quando é requerido este serviço, só é necessário passar por parâmetro o nome do ficheiro, não havendo qualquer necessidade de descrever o caminho completo ou a sua extensão, visto que a solução só suporta registos de áudio WAV.

Após ter sido feita a sua validação, o mesmo ficheiro é interpretado pela biblioteca pyAudioAnalysis, em que o seu objetivo consiste fazer a segmentação do áudio através da *Speaker Diarization (SD)*. Com o auxílio desta tecnologia, é inicialmente feita uma análise que permite compreender o número de oradores presentes na conversa, onde cada um é identificado pelas suas características únicas, extraídas através do áudio. Assim, e a par da utilização de várias janelas temporais com uma duração de vinte centésimos, é feita uma comparação entre as características destas instâncias e as de cada orador, em que o registo analisado será classificado como pertencente àquele que tiver o maior grau de semelhança. No fim desta análise, é retornado um dicionário que contém todas as associações anteriormente realizadas.

O próximo passo deste processo consiste em dividir o ficheiro lido em vários ficheiros de menor duração, utilizando as associações criadas. Para tal, são normalizados os valores referentes aos intervalos de tempo de cada discurso, ou seja, existe uma unificação dos valores destas instâncias contínuas, pertencentes ao mesmo orador (Figura 5.6). Estes valores indicam à Pydub, a biblioteca de manipulação de áudio utilizada, em que locais devem de ser feitas as divisões do ficheiro original.

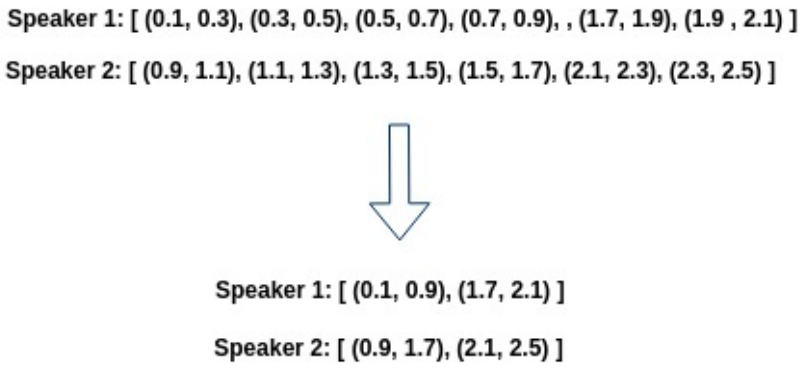

Figura 5.6: Exemplo de normalização dos intervalos temporais obtidos da *Speaker Diarization*

Feitas N divisões de ficheiros, são executadas N *threads* em simultâneo, com a missão de fazer o reconhecimento da fala do áudio com o qual cada uma ficou encarregue. No fim, são recolhidos todos os dados das transcrições feitas, sendo analisados com o classificador de

intenções da Rasa NLU, de modo a que o seu resultado seja armazenado na coleção referente aos diálogos analisados ("Dialogues") e na coleção que contém todo o histórico de diálogos, feitos entre o cliente e o agente ("DialoguesHistory"). Se um diálogo for classificado como pertencente a alguma intenção, é gerada uma resposta por parte do agente, caso contrário, a mensagem é apenas marcada como sendo uma *fallback*. Neste módulo é também criado um ficheiro de *output* com todas as mensagens enviadas pelos participantes da conversa analisada, num formato que contém o tempo inicial e final, o orador e a mensagem de cada diálogo. Este ficheiro é armazenado num servidor interno, podendo ser fornecido um *link* que permita fazer o seu *download*. Na Figura 5.7 é demonstrado um exemplo de um ficheiro de transcrição gerado.

[0.1, 2.7]
0: Bom dia o meu nome é teu carinho em que falsa

[2.7, 12.5]
1: Bom dia eu gostaria de saber em relação a este telemóvel que o meu número que tarifário é que está associado

[12.5, 19.900000000000002]
0: Com certeza então falar com a Dona Maria do Rosário Barreto

[19.900000000000002, 22.500000000000004]
1: Rosário tem 5

[22.500000000000004, 33.7]
0: 96 423 9964 corretamente Sim e tal que está ali Top total 200 megas

[33.7, 50.50000000000001]
1: está o toque do tal dos anos marcas OK assim e ela me interessa a internet não é o pacote

[50.50000000000001, 57.50000000000001]
0: não interessa a internet sim

[57.50000000000001, 62.50000000000001]
1: Ela tem ela tem o telemóvel mesmo só que eles só para receber chamadas e ele

[62.50000000000001, 69.5]
0: Sim é esse é 2 e 80 por semana

Figura 5.7: Ficheiro resultante de uma transcrição

5.7.1 Configurações dos Parâmetros

No STT existem várias situações que podem limitar a qualidade do áudio e consequentemente a sua transcrição. Pormenores como a falta de qualidade da gravação, existência frequente de ruído, chamadas em ambientes agitados, sobreposição dos oradores, sotaques incomuns e má formação das palavras são um obstáculo à obtenção de uma precisão elevada por parte dos sistemas deste tipo. Outro aspeto a ter em conta são os sinónimos fonéticos que, como o próprio nome indica, são palavras ortograficamente diferentes mas que contém uma fonética semelhante. Eis alguns exemplos:

- Sim e cinco;
- Não e mão;
- Em formação e informação;
- Dois e dez;
- Com e cão.

Através de fatores como a dicção, o tom de voz e a velocidade com que é feito o discurso, o resultado final deste conjunto de palavras pode diferir. Assim, e com o objetivo de tentar contornar estas lacunas, o sistema oferece a possibilidade de manipular o áudio original através dos parâmetros velocidade e volume, configurados pelo administrador. No caso destes parâmetros não serem definidos, estes são por defeito, zero para o volume e um para a velocidade, o que representa um ficheiro de áudio sem qualquer tipo de edição.

5.8 Classificação de Intenções

O módulo da classificação de intenções é encarregue de tentar criar correspondências entre as mensagens enviadas pelo cliente e o conjunto de intenções disponíveis na base de conhecimento. Neste tipo de operações existem dois casos possíveis: o agente conversacional não consegue compreender o que foi dito pelo utilizador e informa-o dessa situação, ou então é feita uma correspondência com uma intenção, no qual é gerada uma resposta/ação, configurada especificamente para o caso.

O algoritmo de classificação ocorre através da utilização de um modelo de ML. Este modelo, baseado na aprendizagem supervisionada, é treinado com um conjunto de dados específico, que contém todas as frases de treino, entidades e intenções definidas na base de conhecimento. Na fase de aprendizagem, o modelo tenta fazer, iterativamente, previsões dos dados de treino, de forma a que as mesmas possam ser melhoradas nos passos seguintes. Assim, e no momento da sua execução, este processo pode conter parâmetros como a língua suportada e o valor mínimo de precisão aceitável, capazes de controlar o comportamento do modelo.

Na Rasa NLU, os dados recebidos são treinados por um conjunto sequencial de componentes, através de um processo que dá-se pelo nome de *pipeline*. Neste processo, cada componente contém uma função específica, recebendo como entrada um conjunto de dados próprio, sendo depois gerado um resultado, que servirá como ponto de partida para os outros componentes subsequentes. Os componentes utilizados no *pipeline* são os seguinte:

- Inicialização do Modelo;
- *Tokenization*;
- Extração de Características;
- Extração de Entidades;
- Sinónimos;
- *Duckling*;
- Classificação de Intenções.

5.8.1 Inicialização do Modelo

Na configuração do *pipeline*, existem dois tipos de modelos: modelos pré-treinados ou modelos supervisionados. Os modelos pré-treinados contém categorias de entidades já trabalhadas, referentes a nomes de pessoas, organizações e locais. Este tipo de modelos têm o objetivo de

minimizar o tempo despendido no treino, sendo apropriados para bases de conhecimento que contenham conjuntos de dados pequenos e que usem as entidades já disponibilizadas pelos mesmos. Por outro lado, os modelos supervisionados dedicam mais tempo ao seu treino, de forma a que fiquem aptos para responderem a todas as questões incluídas no domínio do problema aprendido. O modelo utilizado pelo agente conversacional é o modelo supervisionado, visto que o mesmo foi desenvolvido de forma independente e com configurações específicas.

5.8.2 *Tokenization*

O componente referente à *tokenization* segmenta o texto analisado em vários termos. A sua divisão é feita por vários delimitadores como os caracteres especiais e os espaços em branco, onde são aplicadas várias regras específicas, mediante a língua detetada.

5.8.3 Extração de Características

De forma a aplicar os algoritmos de ML num comportamento conversacional, é necessário representar o texto num vetor de características. Assim, para cada resposta recebida é criada uma lista, composta pelo número de repetições de todos os seus termos. Com esta aplicação, é possível calcular a semelhança entre a frase analisada e as frases de treino.

5.8.4 Extração de Entidades

A extração das entidades é feita com o *Conditional Random Field (CRF)*, que é um modelo capaz de reconhecer categorias nas sequências de texto analisadas, mediante as informações obtidas nas correspondências anteriores. Este modelo pode ser considerado como uma cadeia de Markov [94] não direcionada, onde os passos temporais são as palavras e os estados são vistos como as classes das entidades. Desta forma, através de características extraídas das palavras, é calculado um conjunto de valores probabilísticos entre os termos analisados e as categorias de entidades existentes.

5.8.5 Sinónimos

Os dados de treino podem conter vários sinónimos, associados às entidades definidas. Se tal acontecer, este componente assegura que cada conjunto será mapeado para o mesmo valor representativo da entidade a que pertencem.

5.8.6 *Duckling*

O componente de *duckling* tem o objetivo de fazer a unificação das entidades, de forma a retornar os mesmos formatos em categorias como datas, números, distâncias, dinheiro, entre outras. Por exemplo, nas expressões do tipo "25 de junho de 2019", é feita a conversão para um formato do tipo data ("25/06/2019").

5.8.7 Classificação de Intenções

O classificador de intenções é encarregue de treinar o modelo de ML. Este modelo tem como referência o algoritmo StarSpace, cujo objetivo consiste em aprender um conjunto de entidades, representado por várias características discretas, sendo a sua origem um dicionário de tamanho

fixo. Este conjunto, definido na componente da extração de características, permite que o modelo do StarSpace tenha a possibilidade de calcular o grau de semelhança entre as entidades de diferentes tipos [95].

Após a execução do *pipeline*, é gerado um novo modelo de treino. Assim, cada vez que o sistema receber uma nova mensagem, é retornada uma lista de possíveis intenções, acompanhadas do respetivo valor de confiança (Figura 5.8). Adicionalmente, é utilizada uma API dedicada à gestão de contextos, retornando todos os contextos da sessão que se encontra ativa. Com isto, é possível verificar se os contextos de entrada de todas as intenções sugeridas correspondem aos contextos da mesma sessão. No fim, a intenção escolhida para construir a resposta do agente é aquela cujos contextos correspondem aos contextos encontrados na sessão. No caso desta situação acontecer em mais do que uma intenção, é escolhida aquela cujo valor de confiança é maior.


```
{
  "intent": {
    "name": "A0e_Saudações&ApoioCliente&V1.0",
    "confidence": 0.9638760685920715
  },
  "entities": [],
  "intent_ranking": [
    {
      "name": "A0e_Saudações&ApoioCliente&V1.0",
      "confidence": 0.9638760685920715
    },
    {
      "name": "A0e_Segunda Via de Fatura&ApoioCliente&V1.0",
      "confidence": 0.09553061425685883
    },
    {
      "name": "A0e_Dizer Número&ApoioCliente&V1.0",
      "confidence": 0.0
    },
    {
      "name": "A0e_Agradecimentos&ApoioCliente&V1.0",
      "confidence": 0.0
    },
    {
      "name": "A0e_Saber Pin e Puk&ApoioCliente&V1.0",
      "confidence": 0.0
    }
  ],
  "text": "olá",
  "project": "5d208cd3339bdccd5d172cc9",
  "model": "MV1.0"
}
```


Figura 5.8: Exemplo de uma lista de intenções sugeridas

5.9 Identificador de Entidades

O presente bloco é encarregue de procurar entidades existentes no texto. Conforme as mensagens de texto recebidas pelo agente, o objetivo consiste em encontrar termos relevantes, que estejam associados a categorias pré-definidas.

De forma a encontrar os termos desejados, o identificador de entidades auxilia-se de um conjunto de dados criado durante a fase do seu pré-processamento, chamado de *corpus*.

No contexto deste problema, o *corpus* é representado por um conjunto de ficheiros em JSON, criados manualmente, onde cada um simboliza uma categoria de entidades específica, composta pelos seus possíveis valores e respetivos sinónimos. Na próxima figura é apresentada a constituição deste tipo de ficheiros.

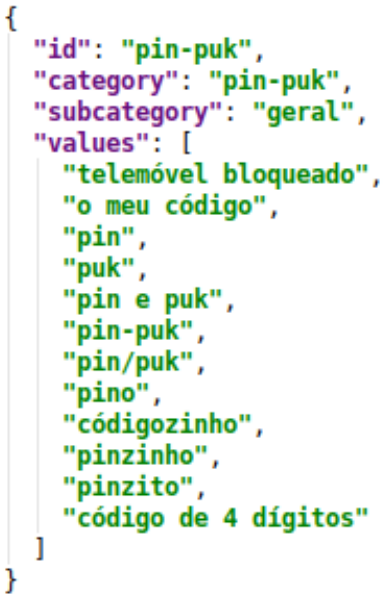

Figura 5.9: Exemplo de uma entidade embutida no *corpus*

Em suma, o algoritmo de pesquisa e identificação das entidades é composto pelos seguintes passos:

1. Manipulação do Texto;
2. *Tokenization*;
3. Remoção das *Stopwords*;
4. *Stemming*;
5. Pesquisa com Contexto;
6. Desambiguação dos Resultados.

5.9.1 Manipulação do Texto

Previamente à implementação do algoritmo de pesquisa, tanto a mensagem recebida como as entidades pré-definidas necessitam de ter o mesmo formato, de modo a que seja possível comparar os seus valores. Como resposta a tal necessidade, todo o texto é convertido para letras minúsculas e são modificadas todas as palavras compostas por justaposição, separando-as em vários termos distintos. Este tipo de manipulações facilita a *tokenization*, fazendo com que seja possível separar os termos da forma desejada.

5.9.2 *Tokenization*

A *tokenization* consiste na separação de uma sequência de palavras em termos individuais, que podem representar palavras, símbolos, palavras-chaves, frases ou outro tipo de símbolos chamados *tokens*. Esta separação é feita mediante a especificação de um elementos textual,

em que neste caso, o elemento separador escolhido é o espaço em branco. Como resultado deste processo, cada frase é compreendida por uma lista de termos individuais, sendo este o formato apropriado para fazer a filtragem das *stopwords*.

5.9.3 Remoção das *Stopwords*

Uma lista de *stopwords* é uma lista de palavras, composta maioritariamente por elementos como pronomes, determinantes, artigos definidos e indefinidos, conjunções, entre outros. Dada a sua elevada utilização na linguagem natural humana, estes termos são considerados pouco relevantes, contudo, podem limitar os resultados obtidos de uma pesquisa, mesmo com a existência de frases semelhantes. Assim, e através da lista de *stopwords* fornecida pelo Snowball Stemmer [96], é feita uma filtragem dos termos anteriores, onde são removidos aqueles que são considerados uma *stopword*. A filtragem deste tipo de palavras traz uma maior tolerância à comparação do texto, fazendo com que duas frases semelhantes sejam consideradas iguais. Na próxima figura é apresentado um exemplo que permite observar as diferenças de uma frase com e sem *stopwords*.

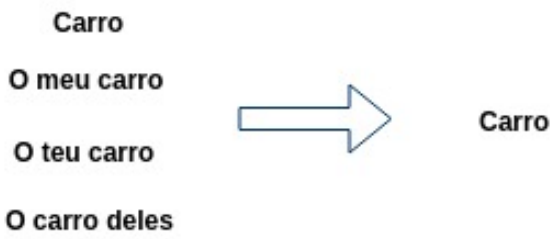

Figura 5.10: Exemplo de remoção das *Stopwords*

5.9.4 *Stemming*

Na pesquisa de texto, a flexão gramatical deve de ser outra característica a ter em conta. Isto é, a pesquisa por um termo no seu singular ou plural, assim como a utilização de uma forma verbal diferente, influenciam o resultado final das correspondências encontradas, o que pode originar novamente a um conjunto de resultados incompleto ou até inexistente, mesmo que duas frases tenham um índice de semelhança elevado. Para contornar este problema, a solução proposta usufrui de um algoritmo de *stemming* para a língua portuguesa, desenvolvido pelo Snowball Stemmer [97], que consiste em remover o prefixo e/ou sufixo de cada palavra, convertendo-a para a sua forma raiz.

5.9.5 Pesquisa com Contexto

Para encontrar as entidades presentes na mensagem enviada pelo cliente, o algoritmo de pesquisa é baseado no modelo das cadeias de Markov [94]. Isto é, em cada termo anteriormente separado e modificado, é utilizado um modelo de contexto finito que permite gerar um conjunto de sub-frases de diferentes ordens. A definição deste tipo de ordem consiste no número de termos que estão exatamente a seguir ao termo atual, ou seja, um modelo de primeira ordem permite criar uma sub-palavra que contém o termo atual e o que está exatamente a seguir, enquanto um modelo de segunda ordem contém três termos: o termo atual e os dois seguintes.

Durante a geração das combinações das várias sub-frases e ordens, cada instância é confrontada com os valores das entidades existentes, contidas num *corpus* pré-criado. No caso de haver alguma correspondência, é adicionado um novo objeto a uma lista de entidades já encontradas, o que representa uma nova associação entre a sub-frase analisada e a categoria correspondente (Figura 5.11).

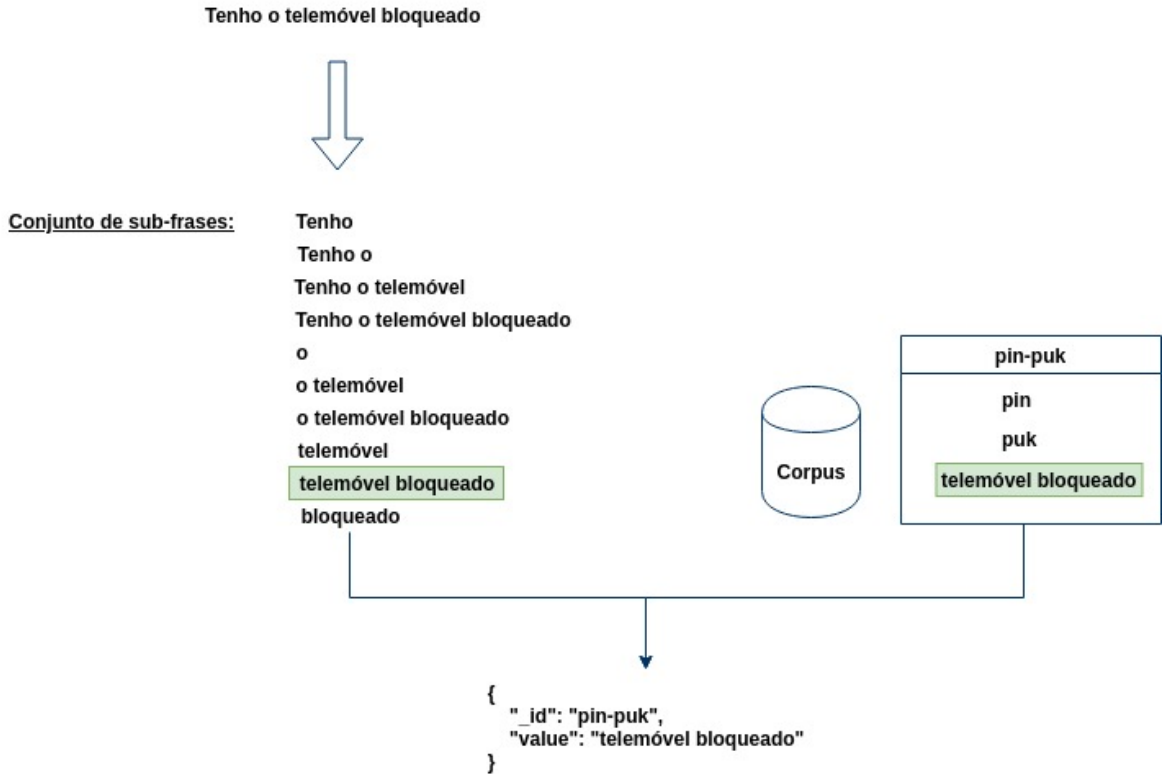

Figura 5.11: Exemplo de aplicação do identificador de entidades

5.9.6 Desambiguação dos Resultados

O resultado proveniente da pesquisa de entidades consiste numa lista que contém todas as associações criadas entre os termos encontrados e as categorias correspondentes. Um dos principais problemas nestas operações é o facto das mesmas palavras serem utilizadas várias vezes durante o seu processo, apesar de enquadrarem em contextos diferentes. Com isto, e devido à existência de valores que são sub-palavras de outras entidades, são criados resultados ambíguos, o que consequentemente originam um conjunto de associações incompreensíveis, repetidas e imprecisas.

A ambiguidade dos resultados pode ocorrer de duas formas: o mesmo valor ser associado a duas categorias diferentes ou uma categoria ser correspondida em vários termos. De forma a evitar estas situações, a solução proposta possui um método que previne a criação de resultados ambíguos, através dos seguintes passos:

1. Para cada termo da lista de associações, são filtradas outras instâncias que contenham

o mesmo valor ou a mesma categoria;
2. No caso dos termos serem iguais, é removido o termo que foi adicionado à lista em primeiro lugar;
3. No caso de as categorias serem iguais, é removida a instância cujo termo contém um contexto menor, ou no caso de serem termos da mesma ordem, a instância cujo nome é menor;
4. Os passos anteriores são novamente realizados até haver apenas uma instância para cada categoria encontrada.

Na próxima figura é apresentado um exemplo do processo de desambiguação das entidades encontradas.

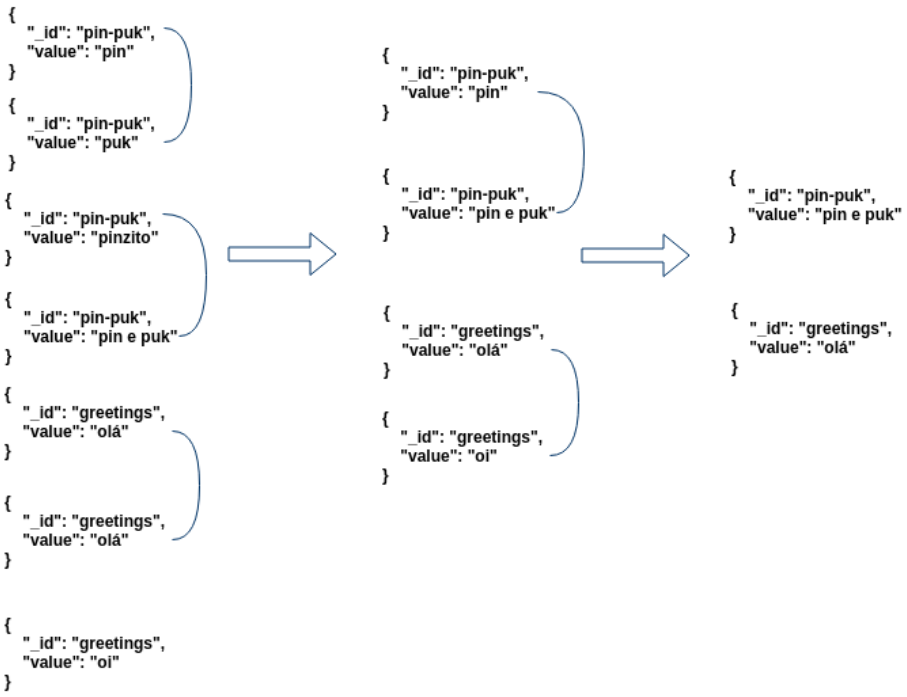

Figura 5.12: Exemplo da desambiguação de entidades

5.10 Construção de Sugestões

O último bloco é dedicado à construção de sugestões que, após a confirmação do administrador, as frases de treino e entidades em causa são adicionada dinamicamente às intenções correspondentes, que fazem parte do conteúdo criado na base de conhecimento do agente.

A classificação das intenções através da Rasa NLU é pouco flexível na produção de resultados, isto é, as respostas enviadas pelo cliente só são reconhecidas se forem praticamente iguais às frases de treino existentes na base de conhecimento. Caso as mensagens recebidas contenham um termo incomum ou uma estrutura gramatical ligeiramente diferente, existe uma grande probabilidade de o agente não compreender o pedido feito, o que leva à necessidade de

construir uma base de conhecimento de grandes proporções. Esta abordagem, não só requer um grande esforço manual e um grande investimento temporal, como também não é garantida a total compreensão da parte do agente após a aplicação desta prática.

Para colmatar as dificuldades existentes, foi criado um módulo de sugestões que pretende sugerir, mediante as entidades reconhecidas, a associação automática de várias frases de treino a intenções contidas na base de conhecimento. Assim, o primeiro passo consiste na recolha dos diálogos, armazenados da coleção "DialogueHistory". Este conjunto de dados contém o histórico de todas as conversações feitas entre o cliente e o agente em causa, de onde são retiradas as mensagens que não foram compreendidas pelo mesmo, ou seja, aquelas cujo tópico corresponde a uma "fallbackIntent" ou "NO_INTENT".

No próximo passo, o módulo identificador de entidades analisa todas as incompreensões ocorridas, e mediante os resultados obtidos, é feita uma comparação entre estes e o conteúdo criado na base de conhecimento do agente. Ou seja, nesta base de conhecimento são analisadas todas as intenções criadas, de onde é retirado o conjunto de entidades utilizado. Estas, após sofrerem algumas alterações no texto, são confrontadas com as entidades anteriormente encontradas pelo módulo identificador. No caso de haver, pelo menos, uma correspondência na interseção destes dois conjuntos, é então criada uma nova sugestão de associação entre a frase analisada e a intenção que contém um conjunto de entidades idêntico. Para além de ser possível adicionar frases de treino às intenções, cada sugestão permite que também sejam criadas entidades e sinónimos extra que foram encontrados no módulo identificador e que ainda não se encontram na respetiva intenção.

No caso de uma intenção não possuir nenhuma entidade, o processo anterior é na mesma aplicado, com a diferença de que, em vez de ser analisado o seu conjunto de entidades, é feita uma comparação com o seu próprio nome. Esta funcionalidade é útil para as intenções que contenham apenas frases de treino simples, como é o caso da "**Saudações**", que consiste num conjunto de frases referentes a boas-vindas.

Através da WebCT, foi criado um *mock*, de forma a que o administrador consiga interagir com as sugestões dadas pelo sistema. Na página referente aos detalhes do agente conversacional, o separador "Suggestions" apresenta uma tabela com todas as sugestões feitas, sendo possível realizar três ações (Figura 5.13):

- **Confirmar**: a sugestão é aceite e a frase de treino é adicionada à intenção proposta, assim como as entidades e os sinónimos extra, no caso destes terem sido detetados;
- **Ignorar**: a sugestão não se enquadra na intenção para a qual foi proposta. O administrador pode removê-la, fazendo com que esta não volte a ser apresentada na tabela de sugestões;
- **Remover**: após a sugestão ter sido aceite, é possível voltar atrás no mesmo passo. Assim, todo o conteúdo que foi anteriormente adicionado é removido da intenção na qual

estava contido. Com este recuo, a instância em causa volta a ser uma mera sugestão.

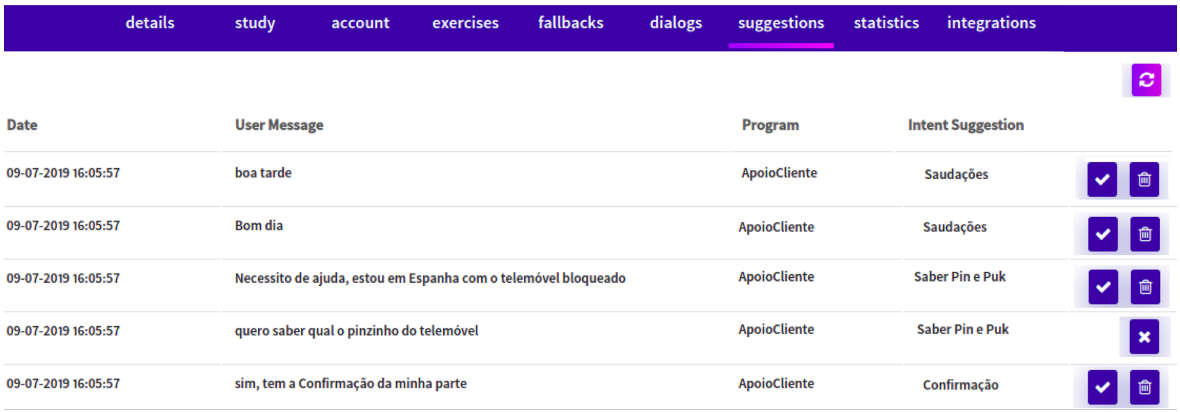

Figura 5.13: *Mock* das sugestões de intenções

5.11 Fluxo de Execução do Agente

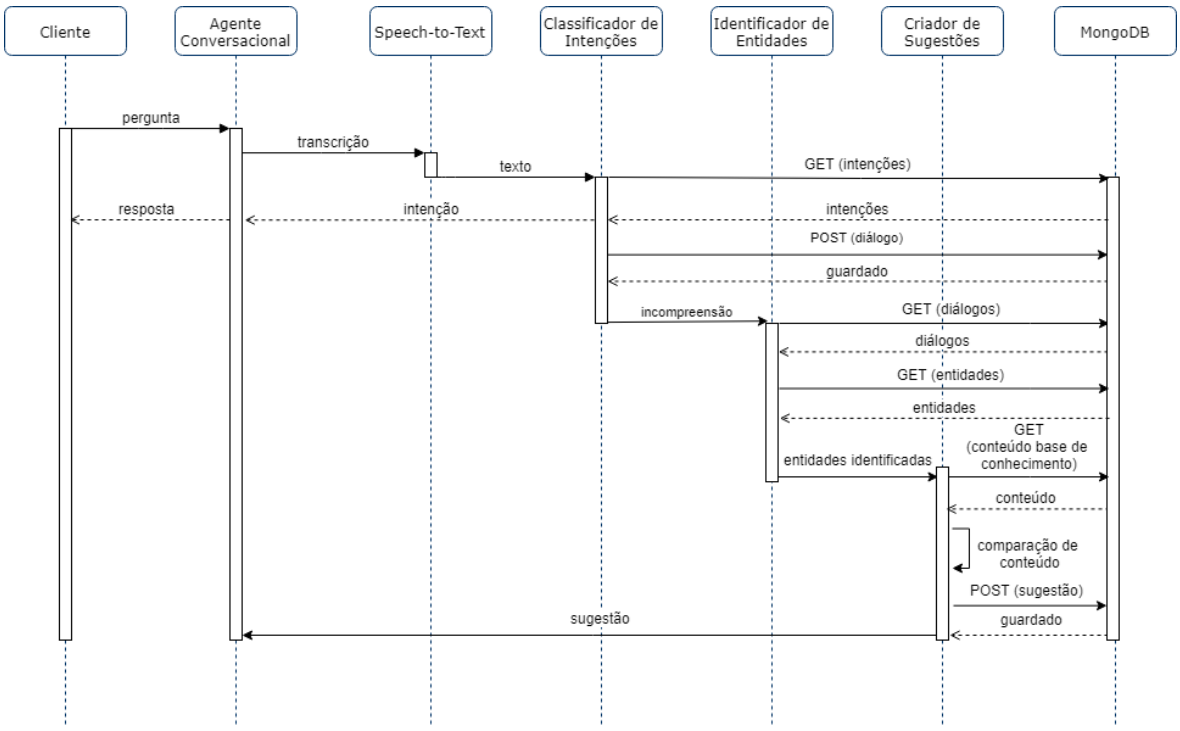

Figura 5.14: Fluxo de Execução do Agente Conversacional

Na Figura 5.14 é apresentado o fluxo de execução do agente conversacional, que é composto pelos seguintes passos:

1. O cliente envia uma pergunta para o agente conversacional;
2. O agente recebe a pergunta por voz e, através do bloco STT, é feita a transcrição para texto;

3. O texto transcrito é enviado para o classificador de intenções. Este bloco, através do modelo de ML treinado, e com o auxílio da Rasa NLU, tenta fazer uma correspondência entre a frase analisada e a intenção definida no conteúdo da base de conhecimento do agente. No caso de ter sido feita alguma classificação, é enviada uma resposta, configurada especificamente para a situação em causa, caso contrário, o agente informa o cliente que não compreendeu o pedido feito. No fim, é adicionada à coleção do histórico de diálogos um novo documento que contém o pedido feito, a intenção classificada e a resposta dada pelo agente;
4. O identificador de entidades é utilizado nos momentos em que o agente não compreendeu o que foi dito pelo cliente. Este bloco, através dos dados recolhidos do histórico de diálogos, analisa todas as frases que não obtiveram nenhuma correspondência com as intenções definidas na base de conhecimento. Assim, para cada instância deste conjunto, o identificador de entidades tenta encontrar palavras-chave e termos relevantes, associando-os a categoria pré-definidas, que se encontram armazenadas na Base de Dados. No fim, são retornadas todas as associações feitas entre as entidades encontradas e as respetivas categorias;
5. O criador de sugestões é utilizado nos casos em que o identificador de entidades conseguiu fazer alguma associação. Neste bloco, o conteúdo da base de conhecimento do agente é comparado com os resultados obtidos no passo anterior. No caso de haver alguma frase e/ou entidade que seja semelhante ao conteúdo da base de conhecimento, e que ainda não esteja implementada na mesma, é então criada uma nova sugestão com o objetivo de realizar essa mesma tarefa de forma dinâmica;
6. Todas as sugestões criadas são apresentadas numa página própria, estando apenas visível para o administrador do agente. Assim, cabe a este interveniente decidir se cada sugestão feita deve de ser integrada na base de conhecimento. Quando uma sugestão é aceite, é gerado um novo modelo de ML, de forma a que o agente consiga compreender o novo conteúdo adicionado.

CAPÍTULO 6

Resultados Obtidos

O presente capítulo é responsável por apresentar um conjunto de resultados que comprovam o cumprimento da solução desenvolvida, perante os desafios propostos. Primeiramente, são descritas as experiências feitas nos vários blocos e os respetivos resultados, de forma a retirar conclusões sobre o seu comportamento. De seguida e em último lugar, é feita uma comparação direta entre o processo manual e o automatizado, em relação aos custos e tempo, o que permite compreender quais são as vantagens em utilizar a solução proposta.

6.1 *Speech-to-Text*

Para testar este bloco, foram analisados os resultados obtidos através de 24 registos de chamadas realizadas. Estas conversas contam com a participação de um operador e um cliente, onde são discutidos vários temas como a faturação, pacotes móveis e de *internet*, *roaming*, planos de tarifários, entre outros. Neste conjunto existe uma variação nos níveis de qualidade, onde para além das diferenças do ruído do ambiente que os rodeia, existem intervenientes que contém uma dicção mais clara e um ritmo mais ponderado do que outros.

Perante a necessidade de ter um valor de referência, com o qual é possível comparar outras configurações, primeiramente foi feita a transcrição manual de todos os registos disponibilizados. Estas transcrições foram construídas num formato idêntico ao do que é gerado pelas transcrições automáticas, onde a precisão, dependente do estado em que se encontra a chamada, consegue alcançar valores muito próximos dos 100%. Assim, e de forma a avaliar a qualidade das transcrições realizadas pelo presente bloco, é feita uma comparação com a transcrição original, utilizando as seguintes medidas:

- **Semelhança do cosseno**: grau de semelhança entre duas fontes de texto, através do cosseno do ângulo entre dois vetores;
- **Coeficiente de Jaccard**: tamanho da interseção de dois conjuntos, dividido pelo tamanho da união dos mesmos;

- **Número de diálogos gerados**: de modo a compreender a eficiência da separação dos diálogos, é também calculada a percentagem do número de divisões feitas numa conversa pelo número de diálogos existentes na transcrição manual.

Medida	Mínimo	Média	Máximo
Semelhança do Cosseno	0,3	0,566	0,778
Coeficiente de Jaccard	0,162	0,329	0,462
Número de Diálogos divididos	3,92%	70,45%	200%
Duração do áudio (segundos)	106	307,583	1141
Duração da Transcrição (segundos)	9,090	35,207	120,229

Tabela 6.1: Valores das transcrições obtidos através das medidas de semelhança e tempo

Na tabela anterior são apresentados os valores mínimos, médios e máximos das medidas de semelhança, perante as transcrições feitas para todos os registos de áudio. São também exibidos dois campos referentes à duração das conversas e à duração da respetiva transcrição, de forma a que seja possível comparar os seus valores.

Em relação à semelhança do cosseno, é possível verificar que, em média, uma transcrição automática contém cerca de 57% do texto de uma transcrição manual. No caso da chamada apresentar uma qualidade razoável, este valor sobe até aos 78%, onde por outro lado, desce acentuadamente para os 30% quando a mesma apresenta muito ruído, várias secções de silêncio ou até uma taxa elevada de sobreposição dos oradores, acompanhada de uma possível má dicção. A mesma diferença acontece no Coeficiente de Jaccard. Apesar de obter valores inferiores, devido ao facto de não contabilizar as repetições das palavras durante um diálogo, a percentagem mínima, média e máxima perante a transcrição original é cerca de 16%, 33% e 46%, respetivamente.

Quanto à divisão feita pela *Speaker Diarization (SD)*, esta conseguiu dividir 70% das vezes do que era suposto. A discrepância entre o valor médio e os seus extremos representa a diferença entre uma chamada aceitável e uma chamada que, ou contém muito ruído (200%) ou que tem o volume muito baixo, dando a perceção que é sempre o mesmo orador a discursar (4%).

No que toca ao tempo despendido, uma transcrição automática demora, em média, apenas 9% da duração do registo de áudio analisado. Este valor pode ser ainda menor no caso de a chamada ser de boa qualidade, como também pode aumentar nos casos em que não é possível transcrever nenhuma palavra. Em situações como estas, o algoritmo do reconhecimento da fala percorre todo o seu conjunto de dados, tentando encontrar uma correspondência entre o sinal de áudio interpretado e os que foram aprendidos pelo seu modelo.

Seguidamente foram testadas algumas hipóteses com os parâmetros relativos ao volume e à velocidade. De modo a comparar o valor médio das medidas de semelhança, obtido através

destes ajustes, foram criadas as seguintes combinações:

- **Combinação 1**: Volume = 0 e Velocidade = 0,8;
- **Combinação 2**: Volume = 0 e Velocidade = 1. Valor por defeito, cujos resultados são apresentados na Tabela 6.1;
- **Combinação 3**: Volume = 0 e Velocidade = 1,2;
- **Combinação 4**: Volume = 3 e Velocidade = 0,8;
- **Combinação 5**: Volume = 3 e Velocidade = 1.

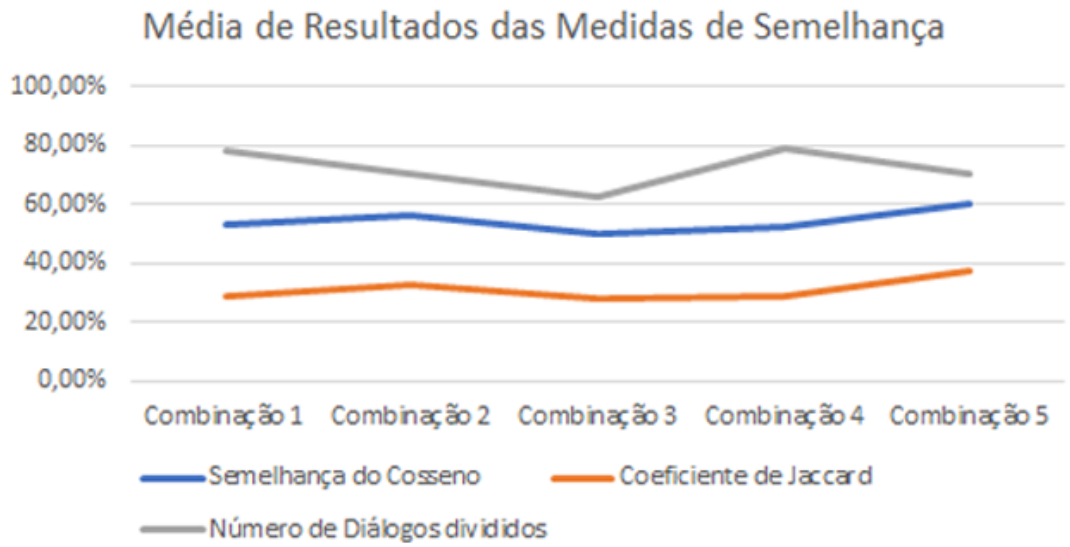

Figura 6.1: Média de Resultados obtidos através das Medidas de Semelhança

Através da figura anterior, é possível verificar que as várias configurações feitas geraram resultados ligeiramente diferentes. Das cinco combinações criadas, a que apresentou melhores resultados é a Combinação 5. Nesta configuração, o volume foi aumentado três vezes mais do que o seu valor original, o que levou a uma melhor compreensão das chamadas cujo áudio não é percetível, devido à baixa ampliação de voz de alguns clientes. Em consequência disso, a mesma obteve um aumento de 0,57 para 0,60 na semelhança do cosseno e de 0,33 para 0,37 no coeficiente de Jaccard.

Contrariamente à combinação anterior, a Combinação 3 foi a que obteve piores resultados. Por norma, uma alteração significativa na velocidade de reprodução resulta num áudio distorcido, fazendo com que não seja possível identificar muitas das palavras ditas pelo orador. Contudo, existem algumas exceções, como por exemplo, as chamadas número onze, vinte e vinte e um do conjunto de registos de áudios analisados. Nestas situações, foi detetado um discurso vagaroso da parte do cliente, o que por defeito origina um conjunto de resultados insatisfatórios. Assim, com a utilização da Combinação 3 nestes casos, os valores da semelhança do cosseno e do coeficiente de Jaccard subiram para o dobro.

Por último, as Combinações 1 e 4 apresentaram um conjunto de valores que, à exceção da Combinação 3, é inferior aos resultados obtidos nas restantes combinações. Apesar disso, as

mesmas realçam-se pelo facto de trabalharem com registos de áudio mais lentos, originando um maior número de diálogos extraídos. Outro pormenor a destacar, é a grande semelhança entre estas duas configurações. Apesar da Combinação 4, em comparação com a Combinação 1, ter feito um aumento ao volume, esta pouco contribuiu para a melhoria dos seus resultados. Através deste caso, conclui-se que a mudança da velocidade de reprodução tem maior impacto nos resultados do que a mudança do volume.

6.2 Identificação de Entidades

O bloco identificador de entidades é composto por 701 valores distintos, organizados por 34 subcategorias, agregadas em 16 categorias principais, das quais se realçam:

- **alterar titular**: alteração de pacotes, tarifários ou contrato;
- **apoio técnico**: necessidade de ajuda técnica, derivada de problemas existente na televisão, *internet* ou telefone;
- **assistente**: pedido de ajuda a um assistente, de forma a solucionar o problema em causa;
- **comercial**: situações como ofertas comerciais ou a compra de algum telemóvel;
- **confirmação**: acordo na proposta feita ou afirmação de algo;
- **desativação**: desativação de tarifário ou desvinculação de contrato;
- **faturação**: situação referentes a faturas, mensalidades e planos;
- **saudações**: boas-vindas e cumprimentos;
- **local**: continentes, países e respetivas capitais, assim como distritos e concelhos portugueses;
- **negação**: desacordo na proposta feita ou não confirmação de algo;
- **NIF**: termos referentes ao número de identificação fiscal, ou por outras palavras, número de contribuinte;
- ***pin-puk***: códigos de acesso e de desbloqueio do cartão SIM;
- **saldo**: quantia existente na conta, sendo utilizada em serviços de telecomunicações.

Para compreender os resultados obtidos por este módulo, foram analisados cerca de 450 diálogos dos 900 gerados, através das transcrições de áudio. Este conjunto corresponde às respostas dadas pelos clientes, com as quais a solução desenvolvida trabalha.

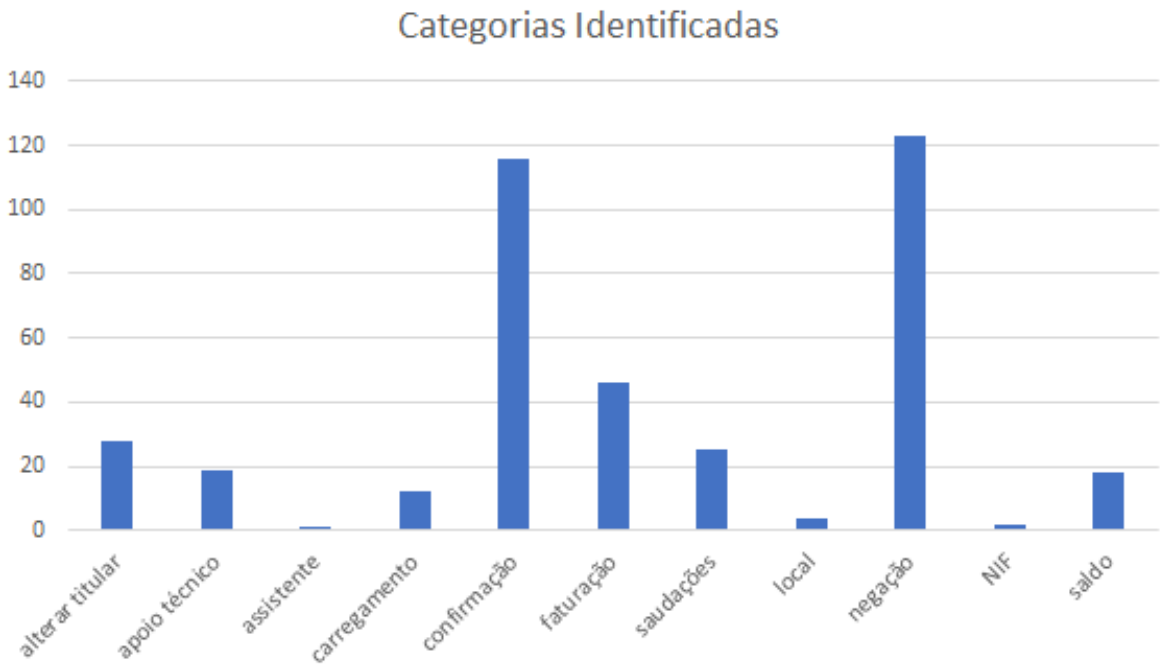

Figura 6.2: Categorias de Entidades reconhecidas no Identificador de Entidades

Através da Figura 6.2, é possível verificar que das 394 identificações feitas, mais de metade correspondeu a confirmações ou negações. Este era um resultado esperado, visto que durante uma conversa, independentemente do seu tema, são frequentemente utilizadas palavras como "sim" e "não", "ok", "concordo" e "não concordo", "está bem" e "está mal", entre outras.

Como grande parte dos registos de áudio analisados debruçam-se em casos de uso como a mudança de tarifário, segundas vias e dúvidas nas faturas, perguntas referentes ao saldo em cartão e o reclamações sobre avarias nos aparelhos, as categorias de faturação, alteração de titular, saldo, carregamentos e apoio técnico, foram as que obtiveram o maior número de identificações, a seguir às confirmações, negações e saudações.

Por último, restam as categorias NIF, assistente e local. As duas primeiras costumam ser categorias complementares a outras mais utilizadas, como por exemplo, a faturação envolve dados de pagamento como o valor, entidade, referência e NIF, enquanto que no apoio técnico, o profissional encarregue de solucionar uma avaria é chamado de assistente. Quanto aos locais e por questões de privacidade, são raros os clientes que mencionam a sua localização atual.

6.3 Processo Manual e Processo Automático

Um dos principais motivos para o desenvolvimento desta solução é a necessidade de reduzir o processo manual da definição de um assistente virtual e das suas bases de conhecimento, compostas por várias intenções e frases de treino. Com a abordagem manual, sempre que o assistente não compreender algo, o conteúdo em análise precisa de ser adicionado à base de conhecimento, de forma a que seja suportado pelo mesmo. Todo este trabalho repetitivo

requer esforço, tempo e custos, o que leva a que seja algo difícil de manter e sobretudo evoluir. Desta forma, a presente secção faz uma comparação entre os tempos despendidos no processo manual e no processo automático.

Para comparar os diferentes tempos de execução, suponha-se que se pretende criar uma base de conhecimento, constituída por um conjunto de cinco intenções, provenientes de diálogos presentes num ficheiro de áudio, com a duração de cinco minutos. No fim, pretende-se adicionar mais três frases de treino ao conjunto de dados, de forma a enriquecer a base de conhecimento. Assim, as diferenças entre os dois processos podem ser comparadas nas seguintes tarefas:

- ***Speech-to-Text***: fazer uma transcrição manual de um ficheiro com esta duração demora, aproximadamente, 45 minutos. Quanto à transcrição automática, como foi dito anteriormente, a mesma demora apenas 9% do tempo total da conversa original, que neste caso corresponde a 27 segundos;
- **Criação da base de conhecimento (Processo de Aprendizagem)**: cada intenção contém, em média, cerca de dez frases de treino. Ou seja, para este caso específico são necessárias cinquenta instâncias. Seguindo a abordagem manual, cada frase de treino demora cerca de cinco minutos, passando por vários processos desde o seu planeamento, a criação de entidades e a sua conceção. No final, o tempo total despendido pelo administrador é cerca de 250 minutos. Com a abordagem automática, analisar uma frase e adicioná-la ao conjunto de dados existente, demora aproximadamente cinco segundos, o que perfaz um total de quatro minutos e dez segundos;
- **Adição de conteúdo (Processo de Classificação)**: no processo manual, adicionar uma nova frase de treino à base de conhecimento demora cerca de quinze minutos, onde são observadas todas as intenções e frases de treino já existentes, associando a nova instância à categoria mais apropriada. Sendo esta tarefa feita três vezes, o tempo estimado ronda os 45 minutos. No processo automático, aceitar uma sugestão dada pelo sistema e adicionar a frase de treino dinamicamente demora cerca de cinco segundos, resultando num total de quinze segundos.

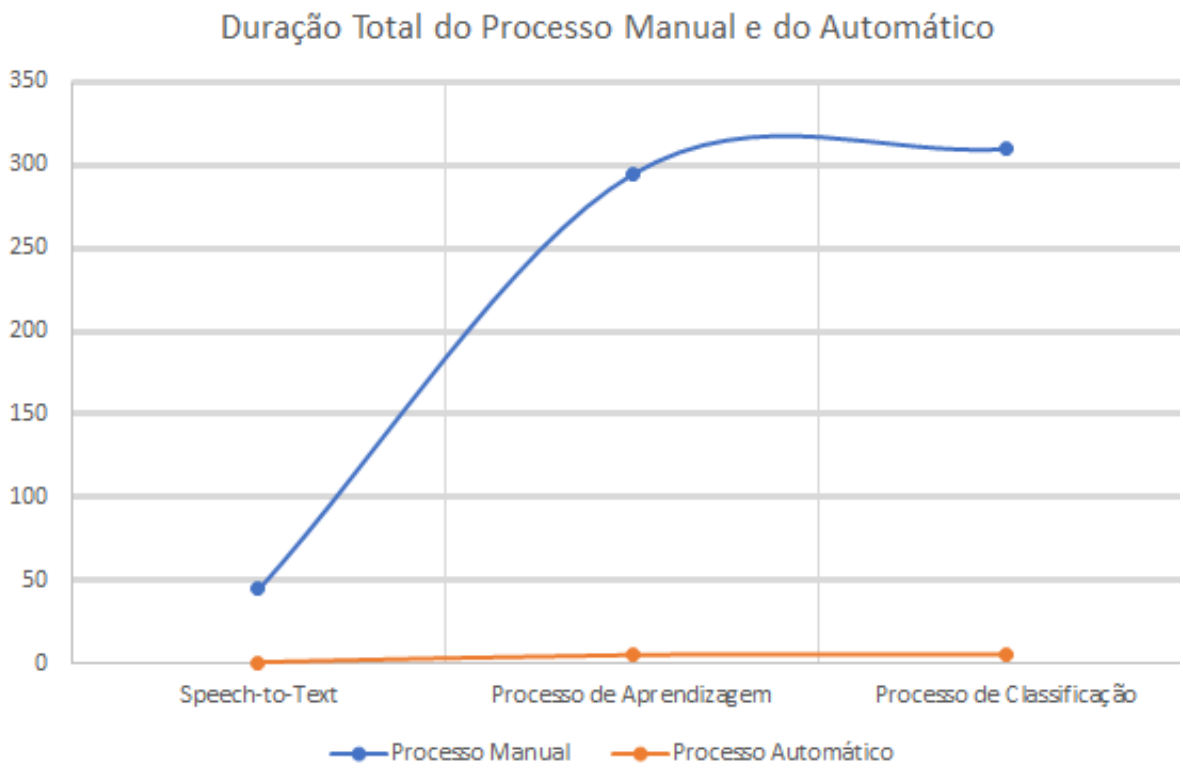

Figura 6.3: Duração total do Processo Manual e do Processo Automático

A figura anterior apresenta a diferença do tempo gasto pelas duas abordagens, ao executar várias tarefas. Através da mesma, verifica-se que, enquanto o processo manual demora mais de cinco horas a ser concluído, o processo automático não necessita sequer de cinco minutos. Com esta discrepância, é fácil de concluir que o método mais rápido é o processo automático. Este não só reduz em 98% o tempo despendido na abordagem manual, como também economiza recursos e custos. É uma abordagem simples, direta e eficaz que se foca num fluxo de execução capaz de criar conteúdo de uma forma rápida e iterativa, sem quaisquer tipos de transtornos ou imprevistos para o cliente.

CAPÍTULO 7

Conclusões e Trabalho Futuro

O presente e último capítulo desta dissertação tem o objetivo de mencionar algumas considerações finais sobre o trabalho desenvolvido. Primeiramente são feitas algumas conclusões, seguidas da descrição dos problemas encontrados durante a sua implementação. Por último, é enumerado um conjunto de ideias a ter em conta para o trabalho a realizar num futuro próximo.

7.1 Conclusões

O objetivo desta dissertação consistiu em analisar, definir e implementar os melhores métodos para fazer a análise de texto e respetivo reconhecimento de entidades. Juntamente com a possibilidade de desenvolver um módulo de transcrições de áudio, estipulou-se que a solução proposta conseguisse criar um novo fluxo de execução para os assistentes virtuais, com o intuito de adicionar dinamicamente diálogos à sua base de conhecimento.

Em resposta a estas necessidades, foram desenvolvidos dois subsistemas. O primeiro, com o auxílio da Bot School para a construção de um agente conversacional, consistiu na recolha do histórico de diálogos entre o agente e o utilizador. Deste conjunto, todas as respostas incompreendidas foram analisadas pelo identificador de entidades, onde foram aplicados alguns métodos de NLP, que permitiram manipular o texto e fazer o posterior reconhecimento dos seus termos. Mediante os resultados obtidos neste bloco, foi também possível criar um mecanismo que gerou um conjunto de sugestões para o administrador, de forma a que fossem adicionadas entidades, variáveis e frases de treino dinamicamente à base de conhecimento do agente.

O segundo subsistema ficou encarregue da converter áudio para texto, seguindo a metodologia do STT. Através do reconhecimento da fala e da aplicação da *Speaker Diarization (SD)*, foi possível compreender o discurso do cliente e até fazer transcrições de ficheiros de áudio, dividindo o ficheiro original em várias partições, de modo a que fosse possível associar a

frase reconhecida ao seu emissor. A implementação deste módulo não só substituiu o processo manual, como também foi utilizada para adicionar novo conteúdo à base de conhecimento com a qual o agente trabalhou.

Foi possível verificar que a solução proposta, de uma forma inovadora e completamente autónoma, foi capaz de analisar, criar e gerir conteúdo, o que permitiu não só ter uma base de conhecimento mais rica e abrangente, como também resultou numa melhor experiência para o cliente, dada a considerável redução de incompreensões geradas pelo agente. Com esta abordagem, foi possível manter um sistema capaz de realizar as mais diversas tarefas, com a vantagem de conter métodos supervisionados e não-supervisionados, o que leva a uma mistura entre a eficácia e a *performance*.

7.2 Problemas Encontrados

Durante o desenvolvimento do sistema, foram encontradas algumas dificuldades, assim como o surgimento de outros problemas.

Devido aos poucos ficheiros de áudio de teste disponíveis, não foi possível identificar vários termos de categorias de interesse como o *pin* e *puk* ou situações referentes à segunda via do cartão. Para além disso, muitas das chamadas apresentavam pouca qualidade, contendo bastante ruído, sobreposição entre os oradores ou até incompreensão no discurso feito pelos mesmos. Como tal, houve conversões de áudio para texto que ficaram aquém do esperado, havendo casos em que não foi feita a correta divisão e transcrição dos diálogos. De todos os desafios encontrados, este foi o que ofereceu maiores dificuldades.

A remoção de entidades anteriormente adicionadas pelo módulo de sugestões constituiu outro desafio. Perante o painel de sugestões apresentado, o administrador pode confirmar a adição de conteúdo à base de conhecimento do agente. No caso do identificador de entidades ter encontrado entidades e sinónimos extra que correspondam ao domínio do problemas e que ainda não se encontram embutidos na base de conhecimento, a confirmação da sugestão faz com que os mesmos sejam adicionados a tal conjunto de dados. Assim, esta funcionalidade permitiu reduzir a quantidade de sugestões feitas, de modo a que todo o conteúdo desejado fosse agrupado num só componente. Por outro lado, fazer a inversão deste processo não foi muito trivial. Ou seja, supondo que o administrador, em vez de confirmar uma sugestão, pretende remover o conteúdo adicionado dinamicamente pela mesma, só é possível fazê-lo com a frase de treino correspondente à sugestão eliminada, mantendo intacto o conteúdo extra adicionado. Tal situação deve-se ao facto de, para além de não ser possível saber se estas entidades estão a ser utilizadas noutras frases de treino, o modelo de dados construído segue a estrutura exigida pelo Dialogflow, que por sua vez requer um formato bastante restrito para a interpretação do conteúdo gerado.

7.3 Trabalho Futuro

A criação desta solução abriu vários caminhos de possíveis melhorias e investimentos nos mais diversos ramos.

As áreas que necessitam de maior atenção são o STT e a SD. Feito um maior estudo sobre o seu funcionamento e técnicas, pretende-se não só melhorar a qualidade global das transcrições, como também evoluir o processo da divisão dos diálogos feitos, através do reajuste dos intervalos temporais onde cada orador discursa. Á semelhança do volume e da velocidade, existe também a hipótese de criar mais parâmetros de configuração que permitam uma manipulação mais profunda do áudio original. Por último, seria interessante criar um componente que permitisse analisar a qualidade da chamada analisada. Tal funcionalidade ajudaria, por exemplo, a saber quais são as chamadas que não têm conteúdo relevante para o domínio em causa, havendo a possibilidade de serem rejeitadas.

Pretende-se que o identificador de entidades seja enriquecido. Apesar de este já conter um conjunto de dados considerável, muitos valores são genéricos, sendo preciso um refinamento que permita obter uma classificação mais específica e de maior qualidade. O objetivo deste bloco passará por ser um agregador principal de todos os dados do sistema, armazenando toda a informação gerada através das APIs externas.

Por último, está planeada uma integração com o BERT. O BERT é um modelo pré-treinado e bastante poderoso que pode ser utilizado em várias áreas de aplicação, desde a análise e classificação de texto e a gestão de contextos, até funcionalidades como a utilização de *Frequently Asked Questions (FAQ)s*, criação de sinónimos e ajuda na identificação de entidades. Este modelo, focado na aprendizagem não-supervisionada, contém um conhecimento profundo que se reflete na redução do conjunto de dados de treino, na otimização dos seus parâmetros de classificação e na possibilidade de trabalhar com o componente da análise sentimental, que associado ao módulo das transcrições, permite classificar o grau de satisfação de um cliente numa chamada.

Referências

[1] Google, *Google Assistant - Just Say "Hey Google" and Make Google Do It*. URL: https://assistant.google.com/ (acedido em 29/11/2018).

[2] Apple, *Siri*. URL: https://www.apple.com/siri/ (acedido em 29/11/2018).

[3] Amazon, *Amazon Alexa*. URL: https://alexa.amazon.com/spa/index.html%7B%5C#%7Dwelcome (acedido em 29/11/2018).

[4] *Personal Digital Assistant - Cortana Home Assistant - Microsoft*. URL: https://www.microsoft.com/en-us/cortana (acedido em 16/06/2019).

[5] Oxford, *Definition of communication in English*. URL: https://en.oxforddictionaries.com/definition/communication (acedido em 31/03/2019).

[6] M. T. Tausif, S. Chowdhury, M. S. Hawlader, M. Hasanuzzaman e H. Heickal, «Deep Learning Based Bangla Speech-to-Text Conversion», em *2018 5th International Conference on Computational Science/ Intelligence and Applied Informatics (CSII)*, IEEE, jul. de 2018, pp. 49–54, ISBN: 978-1-5386-7875-6. DOI: 10.1109/CSII.2018.00016. URL: https://ieeexplore.ieee.org/document/8457100/.

[7] Y. H. Ghadage e S. D. Shelke, «Speech to text conversion for multilingual languages», em *2016 International Conference on Communication and Signal Processing (ICCSP)*, IEEE, abr. de 2016, pp. 0236–0240, ISBN: 978-1-5090-0396-9. DOI: 10.1109/ICCSP.2016.7754130. URL: http://ieeexplore.ieee.org/document/7754130/.

[8] R. Sultana e R. Palit, «A survey on Bengali speech-to-text recognition techniques», em *2014 9th International Forum on Strategic Technology (IFOST)*, IEEE, out. de 2014, pp. 26–29, ISBN: 978-1-4799-6062-0. DOI: 10.1109/IFOST.2014.6991064. URL: http://ieeexplore.ieee.org/document/6991064/.

[9] M. P. Kesarkar e P. Rao, «Feature Extraction for Speech Recognition», rel. téc., 2003.

[10] P. Verma e P. K. Das, «i-Vectors in speech processing applications: a survey», *International Journal of Speech Technology*, vol. 18, n.º 4, pp. 529–546, dez. de 2015, ISSN: 1381-2416. DOI: 10.1007/s10772-015-9295-3. URL: http://link.springer.com/10.1007/s10772-015-9295-3.

[11] H. Bredin, «pyannote.metrics: A Toolkit for Reproducible Evaluation, Diagnostic, and Error Analysis of Speaker Diarization Systems», em *Interspeech 2017*, ISCA: ISCA, ago. de 2017, pp. 3587–3591. DOI: 10.21437/Interspeech.2017-411. URL: http://www.isca-speech.org/archive/Interspeech%7B%5C_%7D2017/abstracts/0411.html.

[12] O. Galibert, «Methodologies for the evaluation of speaker diarization and automatic speech recognition in the presence of overlapping speech», 2013. URL: https://www.semanticscholar.org/paper/Methodologies-for-the-evaluation-of-speaker-and-in-Galibert/0db8de0d040a9dd5db3d5229fda53f3a1c41e8de.

[13] J. Basu, S. Khan, R. Roy, M. Pal, T. Basu, M. S. Bepari e T. K. Basu, «An overview of speaker diarization: Approaches, resources and challenges», em *2016 Conference of The Oriental Chapter of International Committee for Coordination and Standardization of Speech Databases and Assessment Techniques (O-COCOSDA)*, IEEE, out. de 2016, pp. 166–171, ISBN: 978-1-5090-3516-8. DOI: 10.1109/ICSDA.2016.7919005. URL: http://ieeexplore.ieee.org/document/7919005/.

[14] S. H. Yella e H. Bourlard, «Improved overlap speech diarization of meeting recordings using long-term conversational features», em *2013 IEEE International Conference on Acoustics, Speech and Signal*

Processing, IEEE, mai. de 2013, pp. 7746–7750, ISBN: 978-1-4799-0356-6. DOI: 10.1109/ICASSP.2013.6639171. URL: http://ieeexplore.ieee.org/document/6639171/.

[15] S. Khan, J. Basu, M. Pal, R. Roy e M. S. Bepari, «Multilingual conversational telephony speech corpus creation for real world speaker diarization and recognition», em *2016 Conference of The Oriental Chapter of International Committee for Coordination and Standardization of Speech Databases and Assessment Techniques (O-COCOSDA)*, IEEE, out. de 2016, pp. 177–182, ISBN: 978-1-5090-3516-8. DOI: 10.1109/ICSDA.2016.7919007. URL: http://ieeexplore.ieee.org/document/7919007/.

[16] *ALIZÉ opensource speaker recognition*. URL: https://alize.univ-avignon.fr/ (acedido em 06/05/2019).

[17] J.-F. Bonastre, F. Wils e S. Meignier, «ALIZE, a free toolkit for speaker recognition», em *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, vol. 1, IEEE, pp. 737–740, ISBN: 0-7803-8874-7. DOI: 10.1109/ICASSP.2005.1415219. URL: http://ieeexplore.ieee.org/document/1415219/.

[18] A. Zhang, Q. Wang, Z. Zhu, J. Paisley e C. Wang, «FULLY SUPERVISED SPEAKER DIARIZATION», rel. téc., 2018. arXiv: 1810.04719v7. URL: https://github.com/google/uis-rnn.

[19] *Google AI Blog: Accurate Online Speaker Diarization with Supervised Learning*. URL: https://ai.googleblog.com/2018/11/accurate-online-speaker-diarization.html (acedido em 07/05/2019).

[20] *Pydub by jiaaro*. URL: http://pydub.com/ (acedido em 08/05/2019).

[21] T. Giannakopoulos, «pyAudioAnalysis: An Open-Source Python Library for Audio Signal Analysis», *PLOS ONE*, vol. 10, n.º 12, G. Pavan, ed., e0144610, dez. de 2015, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0144610. URL: http://dx.plos.org/10.1371/journal.pone.0144610.

[22] S. R. Eddy, «Profile hidden Markov models», *Bioinformatics*, vol. 14, n.º 9, pp. 755–763, out. de 1998, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/14.9.755. URL: https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/14.9.755.

[23] S. Gupta, *Overview of Text Similarity Metrics in Python*, 2018. URL: https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50 (acedido em 03/07/2019).

[24] Z. S. Abdallah, M. Carman e G. Haffari, «Multi-domain evaluation framework for named entity recognition tools», *Computer Speech & Language*, vol. 43, pp. 34–55, mai. de 2017, ISSN: 08852308. DOI: 10.1016/j.csl.2016.10.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S0885230815300504.

[25] C. C. Xavier e M. Souza, «Extraction and Classification of Semantic Data from Twitter», 2018. URL: dl.acm.org/citation.cfm?doid=3243082.3264606.

[26] ChengXiang, Zhai, Sean e Massung, «Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining», University of Illinois at Urbana–Champaign, rel. téc., 2016, pp. 23–33. URL: http://www.morganclaypoolpublishers.com/catalog%7B%5C_%7DOrig/samples/9781970001174%7B%5C_%7Dsample.pdf.

[27] MonkeyLearn, *Text Analysis: the only guide you'll ever need*. URL: https://monkeylearn.com/text-analysis/ (acedido em 08/05/2019).

[28] P. Kenekayoro, «Identifying named entities in academic biographies with supervised learning», *Scientometrics*, vol. 116, n.º 2, pp. 751–765, ago. de 2018, ISSN: 0138-9130. DOI: 10.1007/s11192-018-2797-4. URL: http://link.springer.com/10.1007/s11192-018-2797-4.

[29] I. Korvigo, M. Holmatov, A. Zaikovskii e M. Skoblov, «Putting hands to rest: efficient deep CNN-RNN architecture for chemical named entity recognition with no hand-crafted rules», *Journal of Cheminformatics*, vol. 10, p. 28, 2018. DOI: 10.1186/s13321-018-0280-0. URL: jcheminf.springeropen.com/articles/10.1186/s13321-018-0280-0.

[30] D. Maynard, M. A. Greenwood, I. Roberts, G. Windsor e K. Bontcheva, «Real-time Social Media Analytics through Semantic Annotation and Linked Open Data», em *Proceedings of the ACM Web Science Conference on ZZZ - WebSci '15*, New York, New York, USA: ACM Press, 2015, pp. 1–2, ISBN:

9781450336727. DOI: 10.1145/2786451.2786500. URL: http://dl.acm.org/citation.cfm?doid=2786451.2786500.

[31] G. Sheikhshab, E. Starks, A. Karsan, R. Chiu, A. Sarkar e I. Birol, «GraphNER: Using Corpus Level Similarities and Graph Propagation for Named Entity Recognition», em *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, IEEE, mai. de 2018, pp. 229–238, ISBN: 978-1-5386-5555-9. DOI: 10.1109/IPDPSW.2018.00047. URL: https://ieeexplore.ieee.org/document/8425414/.

[32] R. Leaman e G. Gonzalez, «BANNER: An Executable Survey of Advances in Biomedical Named Entity Recognition», rel. téc. URL: http://banner.sourceforge.net..

[33] M. J. Garbade, *A Simple Introduction to Natural Language Processing*, 2018. URL: https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32 (acedido em 08/05/2019).

[34] Lola.com, *NLP vs. NLU: What's the Difference?*, 2016. URL: https://medium.com/@lola.com/nlp-vs-nlu-whats-the-difference-d91c06780992 (acedido em 04/12/2018).

[35] Rasa, *Training Data Format*. URL: https://rasa.com/docs/nlu/dataformat/ (acedido em 09/05/2019).

[36] A. Gatt e E. Krahmer, «Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation», *Journal of Artificial Intelligence Research*, vol. 61, pp. 65–170, jan. de 2018, ISSN: 1076-9757. DOI: 10.1613/jair.5477. URL: https://jair.org/index.php/jair/article/view/11173.

[37] M. Kaput, *The Beginner's Guide to Using Natural Language Generation to Scale Content Marketing*, 2018. URL: https://www.marketingaiinstitute.com/blog/the-beginners-guide-to-using-natural-language-generation-to-scale-content-marketing (acedido em 09/05/2019).

[38] MarutiTechLabs, *What are the advantages of Natural Language Generation and its impact on Business Intelligence? - Maruti Techlabs*. URL: https://www.marutitech.com/advantages-of-natural-language-generation/ (acedido em 09/05/2019).

[39] F. Chiarello, A. Cimino, G. Fantoni e F. Dell'Orletta, «Automatic users extraction from patents», *World Patent Information*, vol. 54, pp. 28–38, set. de 2018, ISSN: 01722190. DOI: 10.1016/j.wpi.2018.07.006. URL: https://linkinghub.elsevier.com/retrieve/pii/S0172219018300115.

[40] A. Singhal e J. Srivastava, «Generating Semantic Annotations For Research Datasets», em *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14) - WIMS '14*, New York, New York, USA: ACM Press, 2014, pp. 1–11, ISBN: 9781450325387. DOI: 10.1145/2611040.2611056. URL: http://dl.acm.org/citation.cfm?doid=2611040.2611056.

[41] M. C. Phan e A. Sun, «CoNEREL: Collective Information Extraction in News Articles», em *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval - SIGIR '18*, New York, New York, USA: ACM Press, 2018, pp. 1273–1276, ISBN: 9781450356572. DOI: 10.1145/3209978.3210165. URL: http://dl.acm.org/citation.cfm?doid=3209978.3210165.

[42] O. Akhtiamov, M. Sidorov, A. Karpov e W. Minker, «Speech and Text Analysis for Multimodal Addressee Detection in Human-Human-Computer Interaction», 2017. DOI: 10.21437/Interspeech.2017-501. URL: http://dx.doi.org/10.21437/Interspeech.2017-501.

[43] J. Jorge Marcos Raposo, «Open Information Extraction from Dialogue Transcriptions», rel. téc., 2016. URL: https://fenix.tecnico.ulisboa.pt/downloadFile/1689244997256335/Thesis.pdf.

[44] V. Maini e S. Sabri, *Machine Learning for Humans*. 2017.

[45] D. Matos, *Conceitos Fundamentais de Machine Learning*, 2015. URL: http://www.cienciaedados.com/conceitos-fundamentais-de-machine-learning/ (acedido em 27/11/2018).

[46] S. Redmore, *Machine Learning for Natural Language Processing - Lexalytics*, 2019. URL: https://www.lexalytics.com/lexablog/machine-learning-vs-natural-language-processing-part-1 (acedido em 18/05/2019).

[47] D. Karunakaran, *Deep learning series 1: Intro to Deep Learning*, 2018. URL: https://medium.com/intro-to-artificial-intelligence/deep-learning-series-1-intro-to-deep-learning-abb1780ee20 (acedido em 28/11/2018).

[48] *GATE - General Architecture for Text Engineering*. URL: https://gate.ac.uk/ (acedido em 13/05/2019).

[49] H. Cunningham, V. Tablan, A. Roberts e K. Bontcheva, «Getting More Out of Biomedical Documents with GATE's Full Lifecycle Open Source Text Analytics», *PLoS Comput Biol*, vol. 9, n.º 2, p. 1 002 854, 2013. DOI: 10.1371/journal.pcbi.1002854. URL: www.ploscompbiol.org.

[50] *The Stanford Natural Language Processing Group*. URL: https://nlp.stanford.edu/software/CRF-NER.html (acedido em 17/05/2019).

[51] T. Reuters, «Open Calais Upgrade Guide», rel. téc., 2015. URL: http://www.opencalais.com/wp-content/uploads/2016/01/ThomsonReutersOpenCalaisUpgradeGuideR9point1d281215.pdf.

[52] X. Zhou, X. Tao, J. Yong e Z. Yang, «Sentiment analysis on tweets for social events», em *Proceedings of the 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, IEEE, jun. de 2013, pp. 557–562, ISBN: 978-1-4673-6085-2. DOI: 10.1109/CSCWD.2013.6581022. URL: http://ieeexplore.ieee.org/document/6581022/.

[53] *Text Analytics – MeaningCloud text mining solutions*. URL: https://www.meaningcloud.com/ (acedido em 15/05/2019).

[54] Apache Software Foundation, *Apache OpenNLP Developer Documentation*, 2011. URL: https://opennlp.apache.org/docs/1.9.1/manual/opennlp.html (acedido em 16/05/2019).

[55] P. Chandrayan, *A Guide To NLP Implementation Using OpenNLP : Making Machines Speak*, 2017. URL: https://codeburst.io/nlp-implementation-using-java-opennlp-guide-and-examples-80d86b02b5b5 (acedido em 16/05/2019).

[56] *TextRazor - The Natural Language Processing API*. URL: https://www.textrazor.com/ (acedido em 16/05/2019).

[57] S. Bird, E. Klein e E. Loper, *Natural Language Processing with Python*, 1ª ed., J. Steele, ed. 2009. URL: http://www.datascienceassn.org/sites/default/files/Natural%20Language%20Processing%20with%20Python.pdf.

[58] A. Navlani, *Text Analytics for Beginners using NLTK (article) - DataCamp*, 2018. URL: https://www.datacamp.com/community/tutorials/text-analytics-beginners-nltk (acedido em 16/05/2019).

[59] E. Loper e S. Bird, «NLTK: The Natural Language Toolkit», mai. de 2002. arXiv: 0205028 [cs]. URL: http://arxiv.org/abs/cs/0205028.

[60] *spaCy 101: Everything you need to know · spaCy Usage Documentation*. URL: https://spacy.io/usage/spacy-101 (acedido em 16/05/2019).

[61] B. Georgeivanov, *spaCy Tutorial - Complete Guide*, 2018. URL: https://nlpforhackers.io/complete-guide-to-spacy/ (acedido em 16/05/2019).

[62] J. Devlin, M.-W. Chang, K. Lee, K. T. Google e A. I. Language, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding», rel. téc. arXiv: 1810.04805v1. URL: https://github.com/tensorflow/tensor2tensor.

[63] R. Horev, *BERT Explained: State of the art language model for NLP*, 2018. URL: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270 (acedido em 17/05/2019).

[64] *What is a Chatbot and How to Use It for Your Business*, 2018. URL: https://medium.com/swlh/what-is-a-chatbot-and-how-to-use-it-for-your-business-976ec2e0a99f (acedido em 19/05/2019).

[65] M. Schlicht, *The Complete Beginner's Guide To Chatbots – Chatbots Magazine*. URL: https://chatbotsmagazine.com/the-complete-beginner-s-guide-to-chatbots-8280b7b906ca (acedido em 19/05/2019).

[66] *Azure Bot Service Introduction - Bot Service | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-overview-introduction?view=azure-bot-service-4.0 (acedido em 19/05/2019).

[67] M. S. Paul, *How to build a chatbot with Dialog flow | Chapter 1 — Introduction*, 2018. URL: https://medium.com/swlh/how-to-build-a-chatbot-with-dialog-flow-chapter-1-introduction-ab880c3428b5 (acedido em 20/05/2019).

[68] Dialogflow, *Overview*. URL: https://dialogflow.com/docs/intro (acedido em 20/05/2019).

[69] Rasa, *Build contextual chatbots and AI assistants with our open source conversational AI framework*. URL: https://rasa.com/docs/ (acedido em 20/05/2019).

[70] M. Lawnboy, *Building a Chatbot Using Rasa Stack: Intro and Tips*, 2017. URL: https://hackernoon.com/building-a-chatbot-using-rasa-stack-intro-and-tips-c6d1057d8536 (acedido em 20/05/2019).

[71] J. M. Sousa, J. M. Monteiro, A. M. Martinho e S. d. S. Furão, «Virtual Assistants, the future digital interface», *InnovAction2018*, pp. 64–73, 2018.

[72] F. Shahzad, «Modern and Responsive Mobile-enabled Web Applications», *Procedia Computer Science*, vol. 110, pp. 410–415, jan. de 2017, ISSN: 1877-0509. DOI: 10.1016/J.PROCS.2017.06.105. URL: https://www.sciencedirect.com/science/article/pii/S187705091731284X?via%7B%5C%%7D3Dihub.

[73] *Introduction - webcomponents.org*. URL: https://www.webcomponents.org/introduction (acedido em 21/05/2019).

[74] P. Schilp, *Web Components: from zero to hero*, 2019. URL: https://dev.to/thepassle/web-components-from-zero-to-hero-4n4m (acedido em 21/05/2019).

[75] *React – A JavaScript library for building user interfaces*. URL: https://reactjs.org/ (acedido em 21/05/2019).

[76] D. Deutsch, *Understanding MVC Architecture with React*, 2017. URL: https://medium.com/of-all-things-tech-progress/understanding-mvc-architecture-with-react-6cd38e91fefd (acedido em 21/05/2019).

[77] F. Copes, *The React Handbook*, 2019. URL: https://medium.freecodecamp.org/the-react-handbook-b71c27b0a795 (acedido em 21/05/2019).

[78] *Introduction — Vue.js*. URL: https://vuejs.org/v2/guide/ (acedido em 22/05/2019).

[79] Z. M. Luo, *MVC vs. MVVM: How a Website Communicates With Its Data Models*, 2017. URL: https://hackernoon.com/mvc-vs-mvvm-how-a-website-communicates-with-its-data-models-18553877bf7d (acedido em 22/05/2019).

[80] F. Copes, *The Vue Handbook: a thorough introduction to Vue.js*, 2018. URL: https://medium.freecodecamp.org/the-vue-handbook-a-thorough-introduction-to-vue-js-1e86835d8446 (acedido em 22/05/2019).

[81] *Getting Started — Vue.js*. URL: https://012.vuejs.org/guide/index.html (acedido em 22/05/2019).

[82] *Comparison with Other Frameworks — Vue.js*. URL: https://vuejs.org/v2/guide/comparison.html (acedido em 22/05/2019).

[83] M. Butusov, *ReactJS vs Angular5 vs Vue.js - What to choose in 2018?*, 2018. URL: https://blog.techmagic.co/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018/ (acedido em 16/12/2018).

[84] *Angular - Architecture overview*. URL: https://angular.io/guide/architecture (acedido em 22/05/2019).

[85] J. Ballin, *I Finally Understand Static vs. Dynamic Typing and You Will Too!*, 2017. URL: https://hackernoon.com/i-finally-understand-static-vs-dynamic-typing-and-you-will-too-ad0c2bd0acc7 (acedido em 27/06/2019).

[86] *Typescript vs JavaScript: What's the Difference?* URL: https://www.guru99.com/typescript-vs-javascript.html (acedido em 22/05/2019).

[87] *React vs Angular vs Vue.js: A Complete Comparison Guide*, 2018. URL: https://www.spec-india.com/blog/react-vs-angular-vs-vue-js-a-complete-comparison-guide/ (acedido em 22/05/2019).

[88] S. Daityari, *Angular vs React vs Vue: Which Framework to Choose in 2019*, 2019. URL: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/ (acedido em 22/05/2019).

[89] *Introduction to Node.js*. URL: https://nodejs.dev/ (acedido em 23/05/2019).

[90] Node.js Foundation, *Why the Hell Would You Use Node.js*, 2017. URL: https://medium.com/the-node-js-collection/why-the-hell-would-you-use-node-js-4b053b94ab8e (acedido em 23/05/2019).

[91] *Introduction to Node.js*. URL: https://nodejs.dev/ (acedido em 23/05/2019).

[92] *Introduction to MongoDB — MongoDB Manual*. URL: https://docs.mongodb.com/manual/introduction/ (acedido em 23/05/2019).

[93] *Documentation*. URL: https://restheart.org/learn/ (acedido em 23/05/2019).

[94] D. S. Myers, L. Wallin e P. Wikström, «An introduction to Markov chains and their applications within finance», rel. téc., 2017. URL: http://www.math.chalmers.se/Stat/Grundutb/CTH/mve220/1617/redingprojects16-17/IntroMarkovChainsandApplications.pdf.

[95] L. Wu, A. Fisch, S. Chopra, K. Adams, A. Bordes e J. Weston, «StarSpace: Embed All The Things!», set. de 2017. arXiv: 1709.03856. URL: http://arxiv.org/abs/1709.03856.

[96] *A Portuguese stop word list*. URL: https://snowballstem.org/algorithms/portuguese/stop.txt (acedido em 29/05/2019).

[97] *Portuguese stemming algorithm*. URL: https://snowballstem.org/algorithms/portuguese/stemmer.html (acedido em 29/05/2019).