UNIVERSIDADE BEIRA INTERIOR
Engenharia

Construção de um Chat-Bot de suporte a actividades empresariais

Daniel Barata Pereira

Relatório de estágio
Mestrado em Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Nuno Manuel Garcia dos Santos
Co-orientador: Eng. João José Teles Gouveia

Covilhã, Junho de 2021

Agradecimentos

Sem dúvida alguma que este projeto de estágio foi um enorme desafio nestes últimos tempos, foram encontrados imensos obstáculos e imensas dúvidas mas devido ao apoio de todos, tudo isto foi ultrapassado.

Em primeiro lugar, quero agradecer ao meu orientador de estágio na ReadinessIT, Eng. João José Teles Gouveia e ao meu orientador na Universidade da Beira Interior, Professor Doutor Nuno Manuel Garcia dos Santos por todo o apoio, dedicação e orientação dada.

Em segundo lugar, quero agradecer à ReadinessIT pela oportunidade de estágio e pela oportunidade de conseguir aprender coisas novas, tanto a nível empresarial como a nível profissional.

Em terceiro lugar, agradeço de um modo especial, à Diana Rodrigues Dias e à Maria Margarida de Almeida Cautela porque sem elas, não tinha chegado onde cheguei e não me tinha metido nesta grande aventura. Obrigado por todo o apoio e pela boa companhia de umas belas noites de estudo.

Quero também agradecer a todos os meus colegas de curso e colegas de trabalho, que foram ajudando e dando todo o apoio e força necessário para conseguir acabar esta etapa.

Por ultimo, agradeço do fundo do meu coração aos meus pais por tudo o que fizeram por mim, por todo o apoio dado, por toda a ajuda dada, pela força e motivação que me deram para tirar um mestrado, por tudo. Sem eles nada disto tinha sido possível ser concretizado. Um enorme obrigado!

Resumo

Este trabalho foi elaborado no âmbito da unidade curricular de projeto de estágio inserido no mestrado em engenharia informática. O principal objetivo deste trabalho consistiu na criação de um chatbot para um meio empresarial, com a capacidade de responder de forma automática aos clientes, sem que haja a necessidade de haver um funcionário a atender. Atualmente, a tecnologia da comunicação está cada vez mais avançada, havendo cada vez mais a opção de comunicação, quer seja por mensagem, quer seja por aplicações de mensagens. Para facilitar a comunicação, as empresas e instituições desenvolvem e investem em chatbots. Estes interagem com as pessoas através de uma interface quer seja por mensagem quer seja por áudio. Estas aplicações permitem um contacto mais eficaz entre o cliente e a empresa, já que numa primeira fase, o atendimento ao cliente passa a ser automático. Para além disso, representa um menor custo operacional para a empresa.

Palavras-chave

Chatbot, Livechat, mensagens automáticas, respostas automáticas, ajuda empresarial.

Abstract

This project was done within the scope of the curricular unit of the internship project inserted in the master's degree in computer engineering. The main objective of this project was the creation of a chatbot for a business environment, with the ability to answer automatically to customers, without the need for an employee to attend. Nowadays, the communication technology is more and more advanced, with the communication option being increasingly used, either by message or by messaging applications. To facilitate communication, companies and institutions develop and invest in chatbots. They interact with people through an interface that wants to be sent by message or by audio. These applications allow a more effective way of a contact between the customer and the company, since in the first phase, customer service becomes automatic. In addition, it represents a lower operating cost for the company.

Keywords

Chatbot, Livechat, automatic messages, automatic answers, entreprise help.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

UBI	Universidade da Beira Interior
CI/CD	Continuous Integration e Continuous Delivery
UX	User Experience
REST	Representational State Transfer
API	Application Programming Interface
UI	User Interface
MEAN	(MongoDB, ExpressJS, Angular e NodeJS
MERN	(MongoDB, ExpressJS, ReactJS e NodeJS
JSON	JavaScript Object Notation
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
HTML	HyperText Markup Language
JS	JavaScript
GUI	Graphical User Interface
JWT	JSON Web Token
NPM	Node Package Manager
NPX	Node Package Executor
URL	Uniform Resource Locator

Capítulo 1

Introdução

Neste capítulo pretendo dar a conhecer a empresa *Readiness IT*, uma empresa de engenharia de software que, tendo as suas raízes no sector das telecomunicações, atualmente pretende expandir-se para outros sectores de negócio (banca, retalho, saúde, governamental). É uma empresa que tem como principal objetivo desenvolver soluções com ”WOW factor” dentro dos padrões de qualidade que lhe são conhecidas em todas as entregas que faz ao cliente. Falarei do recrutamento de pessoas, como é efetuado, e como serão preparadas para o mercado de trabalho. Posteriormente dou a conhecer os tipos de serviços, os clientes que a procuram e, também, as parcerias da *Readiness IT*.

1.1 Caracterização da empresa

1.1.1 Readiness IT

A *Readiness IT* é uma empresa que se encontra em grande expansão no domínio da transformação digital a um nível global, apoiando as empresas a atingirem os melhores resultados através de soluções de negócios de alta qualidade, que desafiam as necessidades do cliente nos dias de hoje. O design de arquitetura, a integração dos sistemas, a automatização de processos, a experiência do utilizador, a automatização de testes, a gestão de relacionamento com o cliente e a experiência multicanal, são alguns dos requisitos da empresa.

A *Readiness IT*, certificada pelo *TM Forum*, foi fundada em 2015, com um grupo de apenas 25 pessoas altamente qualificadas no setor das telecomunicações e atualmente, conta com mais de 400 pessoas, distribuídas por cinco escritórios. A sede localiza-se no Porto e os escritórios situam-se em Lisboa, Fundão, Santiago do Chile, Lima e Auckland.

1.1.2 Clientes

Um grande fator importante para a empresa *Readiness IT* consiste na entrega de soluções integradoras de sistemas complexos utilizando os seus próprios módulos de software (aceleradores), o que transforma o processo de entrega perante o cliente muito mais eficaz, mantendo a qualidade que lhe é reconhecida pelos seus clientes. A empresa tem o impacto com mais de 50 milhões de utilizadores em todo o mundo, nomeadamente:

- Ericsson;
- Entel;
- Spark;

- Oracle;
- Telefonica;
- Swisscom;
- Claro;
- Hewlett Packard;
- T-Mobile;
- Superbock;
- Base;
- Etisalat;
- Orange;
- BMW;
- CenturyLink;
- Verizon;
- Frontier;
- Veon.

1.1.3 System Ninjas

O centro de competências da *Readiness IT* tem o nome de *System Ninjas*, onde o objectivo é formar pessoas nas diversas competências existentes na empresa afim de os preparar para o mercado de trabalho.

O principal foco destas academias é o recrutamento e a formação de recém-licenciados, integrando-os naquilo que são as competências, metodologias de trabalho e valores da empresa.

Desta forma, é uma oportunidade única para iniciar a carreira profissional, onde existem ainda a possibilidade de representar a empresa em projetos *onshore* com clientes internacionais. *Timeline* das academias criadas até hoje pela empresa:

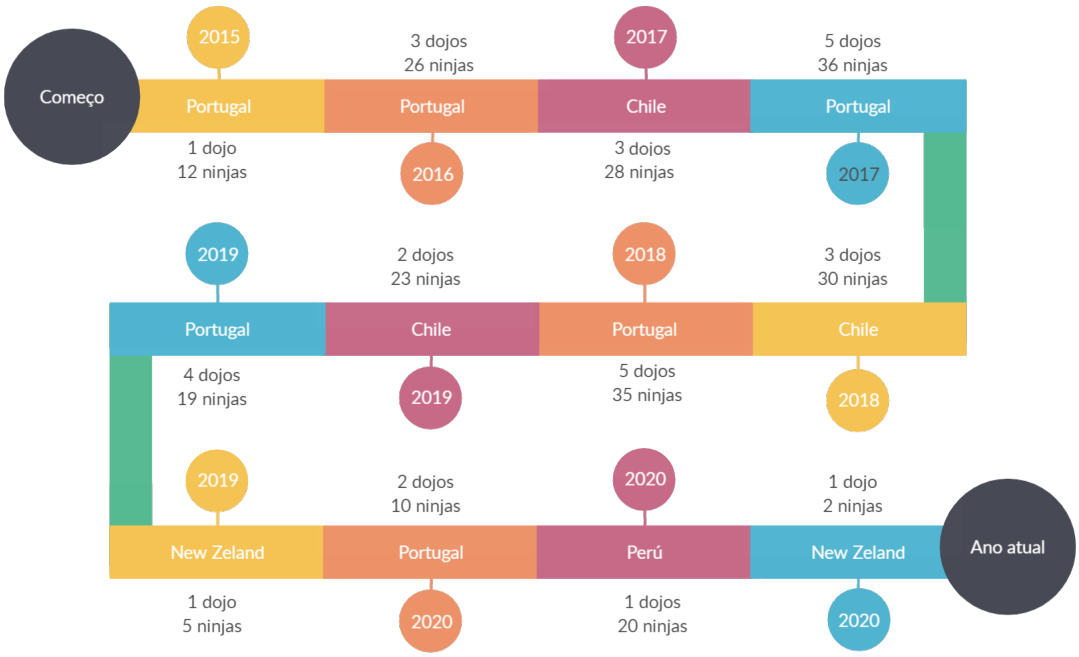

Fig. 1.1: System Ninjas Timeline

Em resumo, passados 5 anos, a empresa recrutou e formou 246 *ninjas*.

1.1.4 Serviços

A empresa *Readiness IT* permite aos clientes o alcance da geração digital. É uma empresa reconhecida mundialmente pelos seus parceiros como uma transformadora digital de sucesso em todo o mundo nos vários setores de negócio.

A sua grande preocupação é elevar a inovação do cliente para o próximo nível, melhorando assim, a produtividade dos negócios e um sucesso a longo prazo, seguindo as melhores práticas do mercado. Cada organização é única e, em conjunto com o cliente, desenvolve-se uma abordagem exclusiva, de forma a construir um programa mais adequado a cada necessidade de negócio. Desde a estratégia inicial até à implementação, o cliente é apoiado na sugestão das tecnologias mais adequadas, nas competências exigidas, no talento da equipa, dando origem a um projeto digital único e personalizado.
A *Readiness IT* garante serviços e profissionais competentes no que se refere às necessidades do cliente. Isto deve-se ao facto de possuir Centros de Excelências e profissionais superiores capazes de satisfazer essas necessidades. A empresa possui equipas qualificadas para os diferentes campos de trabalho que garantem a entrega de projetos, com bastante qualidade e no cronograma do cliente.

- CPQ Catalog - Equipa de catálogo de produtos e serviços, Configuração de todos os produtos, preços e regras de negócios centralizados, automáticos e em tempo real.
- Test Deploy Automation (CI/CD) - Tem por objetivo testar bem os sistemas para a

criação de produtos com qualidade superior para os clientes e negócios. Para tal são utilizadas as melhores ferramentas de teste para assim garantir uma implementação eficiente e efetiva.

- Outsystems Low Coding - Desenvolvimentos referentes a todos os tipos de empresas ou aplicações que se integram facilmente diversos sistemas. Assim, há um rápido desenvolvimento e implementação, mas com maior flexibilidade para adicionar código personalizado, caso seja necessário.
- Digital Experiences (UX/UI/CX/DX) - Soluções que proporcionam experiências significativas e relevantes para os seus utilizadores, desenvolvendo assim, aspetos relativos ao cliente, *branding*, *design*, usabilidade, maquetes e tudo relacionado com a melhoria de experiências de negócios.
- CRM - Responsável pela gestão de clientes
- IT team - Equipa responsável por gerir o CI/CD dos projetos da *Readiness IT* e define os processos de entrega de código juntamente com o cliente
- Backend Services - Consiste em duas arquiteturas de software diferentes, arquitetura monolítica e micro serviços. A arquitetura monolítica possui camadas sobrepostas e são instaladas num só lugar, ou seja, uma única aplicação para controlar tudo. Atualmente a *Readiness IT* possui diferentes módulos, dos quais são: *Order Negocitation*, *Order Manager*, *Billing* e *Charging*. Enquanto a arquitetura de micro serviços, esta é altamente sustentável e testável, fácil de organizar em torno de recursos de negócios e implementado de forma independente. Os módulos existentes comunicam-se universalmente através das interfaces, sendo que cada um realiza a sua tarefa.
- Frontend - Equipa que desenvolve aplicações nativas a partir de tecnologias web (*frameworks*) com base na experiência definida e desenhada pela equipa de UX.

1.1.5 Parcerias

A *Readiness IT* tem parceiros com organizações bem conhecidas para levar a cabo a construção e inovação de negócios a um ritmo mais acelerado. Através de ferramentas tecnológicas é possível entender melhor o que o cliente pretende, desenvolvendo, integrando e transformando o negócio com maior rapidez. A empresa desenvolve e integra soluções digitais, projetadas fundamentalmente para as necessidades de negócio dos clientes.
Parceiros atuais existentes:

- Outsystems
- Vlocity (Salesforce)
- Amazon AWS
- Ericsson

- Kloudville

1.2 Descrição do Problema

A empresa tem necessidade de implementar um *chatbot* com a capacidade de responder de forma automática aos clientes, baseando-se num mecanismo de inteligência artificial capaz de interpretar o que o cliente escreve e decidir que resposta entregar. Isto permite melhorar o processo de apoio ao cliente numa fase prévia e orientar o cliente para o problema real. Esta solução será implementada com o objetivo de ser um *plugin* para qualquer aplicação web, e utilizará a *Framework React*, suportada pela *framework* de UI do *Ant Design*. O *Live Chat* será implementado utilizando a mesma plataforma e reaproveitando os componentes criados para o *chatbot* (desta forma mantemos praticamente transparente a mudança entre o *bot* e a vertente *"live"* na janela de conversação com o cliente). Dado que para construir um *live chat* é necessário ter persistência de dados para gravar as conversações, vai ser implementado um servidor de *back-end* utilizando *expressJS*, onde a comunicação com a aplicação de *front-end* será feita através de *REST APIs*, e construção/ligação uma base de dados em *MongoDB*.

1.3 Organização do Documento

Para mostrar como foi realizado este trabalho, este documento encontra-se estruturado da seguinte forma:

- O primeiro capítulo – **Introdução** – Resume o projeto, mostra o enquadramento do mesmo, a motivação para a sua realização e a organização do documento.
- O segundo capítulo – **Entidade de Acolhimento** – Descreve a entidade de acolhimento
- O terceiro capítulo – **Estado da Arte** – Descreve todas as aplicações já existentes sobre *chatbots*, de como foi feita a pesquisa e qual foi o resultado dos testes das mesmas.
- O quarto capítulo – **Plano de estágio** – Descreve todo o plano de estágio

Capítulo 2

Estado da Arte

2.1 Introdução

Para a realização deste projeto foi necessário fazer um estudo sobre todos os trabalhos, mecanismos e implementações existentes sobre *chatbots*. De facto, a existência de muita informação foi muito útil para este trabalho, tendo conseguido o que precisava para a elaboração do mesmo. Assim sendo, neste capítulo, irá ser descrito todo o processo feito no estudo daquilo que já existe, desde a estratégia de pesquisa, até aos resultados finais.

2.2 Pesquisa

A pesquisa foi realizada no *Google Scholar*, *IEEE Xplore* e *ACM Digital Library*, no mês de Novembro e Dezembro de 2020 pesquisando com as seguintes frases: "*Chatbot*", "*Chatbot Implementation*", "*Chatbot Examples*" e "*Aplications for Chatbot*". O primeiro passo deste estudo foi analisar vários tipos e implementações de *chatbot* e como funcionam. De seguida foi feita uma análise de onde poderíamos aplicar um *chatbot* e por fim foi feita uma compilação de uma lista de *chatbots* já existentes no nosso dia a dia e fazer uma comparação entre elas. No início da pesquisa, foram identificados 142 artigos já com a remoção dos artigos duplicados. Após a leitura do *abstract* e do título, restaram apenas 19 artigos dos quais apenas 6 se identificavam com este estudo. O diagrama de artigos excluídos e incluídos é mostrado na fig. 2.1.

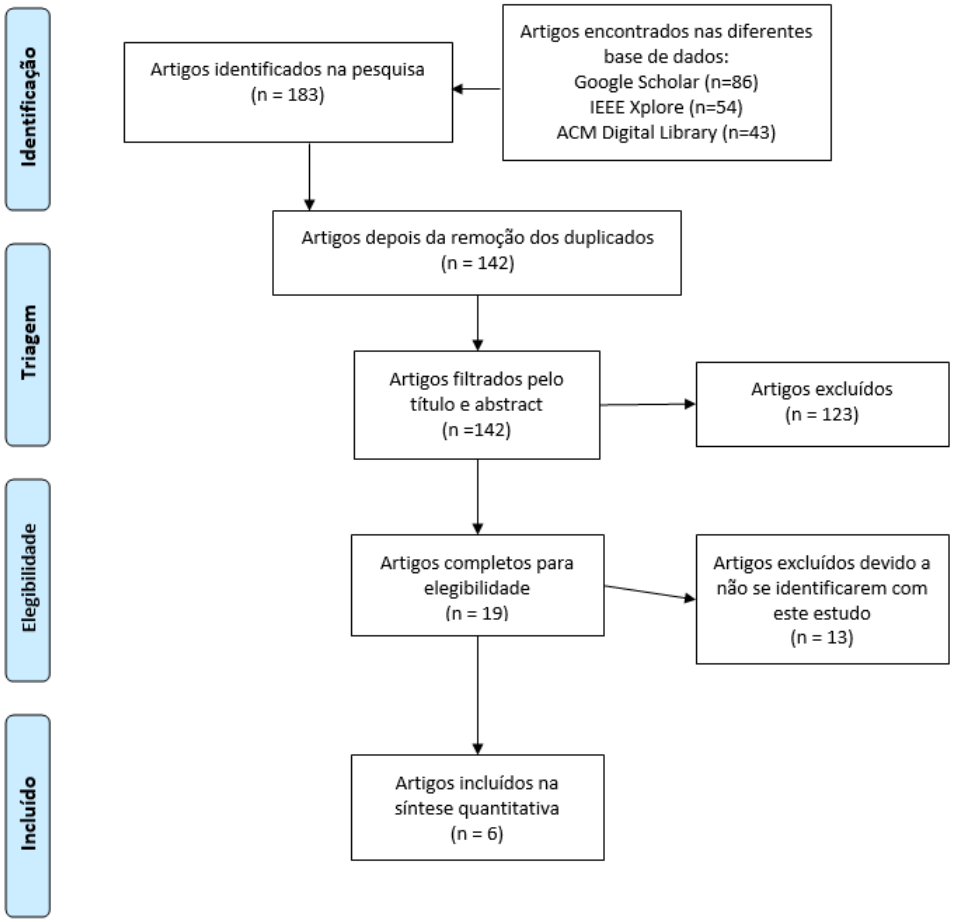

Fig. 2.1: Diagrama de artigos excluídos e incluídos

2.3 Tipos de Chatbot

2.3.1 Pattern Matching

O tipo *Pattern Matching* consiste numa técnica fundamental para projetar *chatbots*, utilizando um conjunto de regras pré-escritas bem como modelos predefinidos para originar a resposta. Pode-se referir que o *chatbot* Eliza foi o primeiro a ser projetado com esta técnica. No princípio, este *chatbot* identificava quais as palavras-chave no texto iniciando da esquerda para a direita. Cada palavra-chave é constituída por um *RANK*. Assim, a *string* de *input* é referida num modelo predefinido. Deste modo, os principais desafios associados a este tipo de abordagem seriam a identificação da palavra-chave mais relevante e a transformação apropriada da regra.
Este tipo de *chatbot* é o mais simples de todos e atualmente está a desaparecer devido às suas limitações. Este programa apenas atende frases literais e não consegue identificar a intenção e o contexto da conversa. [AW20] [Cah17]
Vantagens:

- Disponibilidade de atendimento 24 horas por dia;

- Redução de custos;
- Aprendizagem permanente;
- Maior quantidade de canais de comunicação para uma determinada marca;
- Responde a um conjunto limitado de perguntas e executa tarefas simples.

2.3.2 Parsing

Parsing consiste no processo que leva à quebra da *string* de *input* a fim de revelar a sua estrutura sintática. Em primeiro lugar há que reconhecer adjetivos, artigos e substantivos para formar uma árvore de sintaxe. Quando estamos a analisar é uma ajuda fundamental para validar a estrutura gramatical de uma determinada frase em relação a uma linguagem. Os *chatbots* atuais utilizam técnicas de análise diferentes dos anteriores, técnicas essas que são mais completas e envolvem um processamento de linguagem natural. Estas técnicas de análise abordam a análise sintática, análise semântica e análise pragmática e sintática. A título de exemplo temos o *chatbot* Jabberwacky, que utiliza esta técnica para negócios circunstanciais onde existe um maior controlo de fluxo de conversação. [AW20]

2.3.3 Markov Chain Models

Markov Chain Models descreve a probabilidade de eventos presentes com base no estado dos eventos anteriores. Isto significa ter em consideração a probabilidade com que uma determinada letra ou palavra ocorre num conjunto de dados. Desta forma consegue escolher as palavras mais prováveis para uma resposta. [AW20]
A título de exemplo, o *chatbot* Mega HAL utiliza este método.

2.3.4 Redes Semânticas (Ontologias)

Ontologias consistem numa estrutura hierárquica de conceitos do mundo real, que também são chamados de classes. Quando combinadas com as ontologias, as instâncias de várias classes formam a base do conhecimento. A título de exemplo podemos referir que uma classe ”pão” representa todos os ”pães” e posteriormente são divididos em subclasses, por exemplo ”pão branco” e ”pão integral”. Uma determinada pesquisa pode ser feita sob esta forma e as regras de raciocínio especiais podem implicar novas respostas. [AW20]

2.3.5 AIML

AIML (Artificial Intelligence Markup Language) é um dos avanços tecnológicos no que diz respeito ao desenvolvimento de *chatbots*. É principalmente usado para modelar um diálogo entre um *chatbot* e um humano, onde é seguida a metodologia *stimulus response*. A base de *AIML* consiste no reconhecimento de *Pattern Recognition* e *Matching Techniques*. É fácil de implementar porque está relacionado com *XML* (eXtensive Markup Language) e as *tags* tornam a tarefa de diálogo mais simples. O *Graphmaster* é responsável pela gestão dos padrões de *AIML* que são armazenados. Este fornece uma utilização

bem mais eficiente do tempo e do espaço. Além disso, devido à sua simplicidade, é altamente reutilizável.
A.L.I.C.E foi o primeiro *chatbot* baseado em *AIML*. O modelo de aprendizagem usado no ALICE é supervisionado por alguém, isto é, o *botmaster*. Depois do design inicial de ALICE, muitos outros *chatbots* foram construídos usando *AIML* mas com mais funcionalidades. [SA02] [SB16] [Rah12]

2.3.6 Chatscript

Chatscript é a linguagem de *script* do *chatbot*. Foi desenvolvido por Bruce Wilcox em 2010 com o seu chatbot 'Suzette'. O *chatscript* consiste numa versão improvisada de *AIML*. Aqui as regras são definidas dentro de cada conceito. Os conceitos acabam por ser um conjunto de sinónimos ou palavras semelhantes entre si. As ontologias *Word-net* podem ser combinadas com *chatscript* para assim conseguirem dar respostas mais eficazes. As variáveis também são um conceito importante visto que podem ser utilizadas para armazenamento de informações locais especificas do utilizador, tornando a conversa mais eficaz e natural. [AW20]

2.3.7 Rede Neural

Os *chatbots* baseados em rede neural terminaram com a monotonia de escrever as regras para cada diferente situação. A rede neural pode produzir ou responder desde do zero ou recuperando uma grande quantidade de dados. Para a combinação destes dois foram introduzidas algumas abordagens híbridas. Este tipo de modelo básico foi utilizado pela *SN Computer Science*, em 2020. [AW20] [Cah17]

2.3.8 Rede Neural Recorrente

Rede neural recorrente consiste na capacidade de considerar as conversas anteriores e o contexto enquanto se deseja a criação de uma resposta. Se tivermos apenas em consideração do *input* atual para formar uma resposta, as respostas tornam-se monótonas. A rede neural recorrente *RNN)* permite que o *chatbot* tome o *output* anterior como *input* e chegar a uma resposta mais eficaz, ou seja, permite que os dados persistam ao contrário de uma rede neural normal. As *RNNs* têm sido utilizadas para diversos fins, nomeadamente traduções de linguagem, modelagem de reconhecimento da fala, legenda de imagens, entre outras. [AW20] [Cah17]

2.3.9 Inteligência Artificial

Os *chatbots* possuem uma inteligência artificial mais sofisticada, capaz de identificar a linguagem, o contexto e a intenção do utilizador. Devido ao processamento de Linguagem Natural (PLN), consegue-se entender as diferentes frases e ainda, classificar palavras-chave numa maior amplitude. Conseguem identificar erros à distância e dar respostas coerentes com o perfil e necessidade do utilizador. A inteligência artificial pode ser referida como uma solução para todo o tipo de empresas. [CCM+17] [SB16]

Vantagens:

- São versáteis, podendo ser usados para uma série de atividades;
- São um elo entre o utilizador e os motores de busca;
- Podem resolver situações delicadas;
- Menor custo;
- Otimização do tempo;
- Satisfação do cliente;
- Uso de uma base de dados sólida.

2.4 Onde se pode aplicar um chatbot

2.4.1 E-commerce

Os *chatbots* ficam disponíveis 24 horas por dia. Estes tiram dúvidas sobre produtos, entrega, preços, pagamento, devolução e outros assuntos recorrentes na loja virtual.

2.4.2 Marketing e Vendas

Os *chatbots* são versáteis, onde eles comunicam com os visitantes para melhorar a sua experiência, qualificar os *leads*, prestar atendimento e ainda, vender soluções. Enquanto ajudam o cliente, eles recolhem dados importantes e geram *insights* também importantes no sentido de melhorar as estratégias. Conseguem rapidez e clareza nas mensagens com o cliente. Tem eficiência em:

- Captação de *leads*;
- Verificação das etapas relativas às vendas;
- Realiza um atendimento rápido, melhorando a imagem da empresa perante o cliente;
- Recolhe dados sobre o mercado onde a empresa está inserida;
- Controla os contactos.

2.4.3 Pesquisas

Os *chatbots* conseguem automatizar pesquisas no sentido de impulsionar o seu *marketing*. Através da ferramenta *GrowthBot* é possível pesquisar as últimas tendências e informações de *marketing* e vendas. Além disso, é possível enviar pesquisas para os clientes através dos *chatbots* como o *Wizu*, visto que possui modelos prontos de pesquisa de satisfação.

2.4.4 Atendimento ao Cliente

Os *chatbots* de atendimento ao cliente são dos mais populares. Aqui encontram soluções mais rápidas, seguras e eficientes para as suas questões. Neste sentido, é possível reduzir os custos e aumentar a eficiência dos processos para uma determinada empresa.

2.4.5 Agendamento e Reservas

A especialidade dos *chatbots* são os processos simples e padronizados como o agendamento e reservas. Para tal basta programar o *chatbot* com as opções oferecidas e conectá-lo diretamente à agenda da empresa.

2.4.6 Ensino e Aprendizagem

Relativamente à área da educação, os *chatbots* dão um apoio no que diz respeito ao ensino e aprendizagem nas diversas funcionalidades. A título de exemplo, o *chatbot* pode alertar um aluno para a data de um exame, bem como, indicar conteúdos para estudo e motivar os estudantes.

2.4.7 Medicina

Apesar de existirem algumas controvérsias e dúvidas relativamente à utilização dos *chatbots* na medicina, eles poderão auxiliar e fortalecer o trabalho dos médicos, enfermeiros e outros profissionais de saúde.

Os hospitais cada vez mais seguem a tendência da inteligência artificial uma vez que pode ser útil no auxílio das equipas médicas e dos próprios hospitais no que se refere ao atendimento ao cliente. Neste sentido, é possível atender um paciente sem ter de esperar pela sua vez numa longa fila de pacientes. Assim, a implementação dos *chatbots* nos hospitais permite aos pacientes ter autonomia, rapidez e eficiência quanto ao atendimento. Por sua vez, os profissionais de saúde também conseguem obter maior disponibilidade para se dedicarem a outras tarefas mais complexas e estratégicas.

Um modelo de *chatbot* no hospital é, por exemplo, o *chatbot* para assistência ao paciente. Ele consegue o agendamento da primeira consulta, do seu retorno, do agendamento de exames e do envio do resultado dos mesmos. Temos outros exemplos, nomeadamente, o *chatbot* como conselheiro do paciente, o *chatbot* como consulta médica, entre outros.

A utilização de *chatbots* na medicina permite que as tarefas rotineiras sejam mais eficientes. Uma grande vantagem consiste na sistematização de dados, sem que seja necessário o preenchimento de formulários. Eles tornam-se muito úteis por exemplo, quando surge uma emergência médica. Neste caso, quando o paciente entra numa unidade de saúde, o *chatbot* envia um alerta para a equipa médica que estiver de serviço de emergência e, quando notificados os profissionais, o atendimento é agilizado e o paciente é examinado, de acordo com as suas queixas.

2.5 Live Chat

Atualmente as preferências das pessoas relativamente à comunicação, são cada vez maiores. Os canais a utilizar são muito variados e todos servem propósitos diferentes.
Torna-se muito importante oferecer os mais variados canais de comunicação e também pontos de contacto com os clientes, para que eles possam escolher o que mais interessar.
O *live chat* é um meio de comunicação que permite oferecer aos clientes uma linha de contacto direta para interagir com a empresa. A sua dinâmica da comunicação verbal e escrita difere no formato, nas emoções e na empatia que é possível criar com os clientes. Desta forma, permite que a voz se torne um canal onde o contacto é mais pessoal. Nas áreas financeiras ou da saúde, esse contacto pessoal torna-se importante. Outro aspeto importante é também a rapidez relativamente à transmissão de mensagens.
O *live chat* é um canal de crescimento e tem cada vez mais pessoas a utilizá-lo, tornando-se um dos mais preferidos. Ele pode ser utilizado no sentido de reduzir a pressão sobre os outros canais, absorvendo os clientes com questões mais simples e libertando assim, os agentes para problemas mais complicados.

2.5.1 Live chat versus chatbot

- Atendimento ao cliente - O *live chat* é importante a partir de um atendimento direcionado que garanta uma interação completa e específica para os utilizadores enquanto o *chatbot* é eficiente no que se refere ao esclarecimento de dúvidas e dinamização do atendimento inicial;

- Ensino - O *chatbot* ensina os passos mais simples e básicos de um processo, de maneira mais interativa enquanto o *live chat* entra em ação para dialogar sobre algum assunto de forma mais detalhada;

- Marketing e vendas - Os *chatbots* podem ser implementados no que se refere à realização de campanhas, apresentando mensagens de promoções para o utilizador enquanto o *live chat* é utilizado para iniciar diálogos focados nas vendas, incentivando os clientes relativamente ao envolvimento deles com as promoções.

Comparando o *live chat* com o *chatbot*, ambos podem trazer melhorias relativamente ao atendimento e serviços de uma empresa, resultando assim, numa experiência diferenciada para o cliente. Havendo uma equipa competente e parceiros que ofereçam um bom suporte, o negócio poderá ser vantajoso na competitividade e ainda, levar a um aumento de taxa de fidelidade.
Cada vez mais estas duas ferramentas são implementadas no mercado tendo como finalidade a qualificação do atendimento das empresas bem como a otimização dos processos internos.
O *live chat* e o *chatbot* são tecnologias que levam a uma aproximação entre o cliente e a empresa e também oferecem um serviço para além do convencional.

2.6 Descrição dos Chatbots

Table 2.1: Informação sobre as aplicações *chatbot* resultantes do estudo

Chatbot	Alvo	Empresa	Disponivel 24/7	Apelativo	Live Chat
Siri	Assistente Virtual	Apple	Sim	*****	Não
Cortan	Assistente Virtual	Microsoft	Sim	*****	Não
Google	Assistente Virtual	Google	Sim	*****	Não
Alexa	Assistente Virtual	Amazon	Sim	****	Não
Amtrak	Opções e texto	Amtrak	Sim	***	Não
Amwell	Opções	Amwell	Sim	****	Não
Casper	Opções e texto	Casper	Sim	**	Sim
Heek	Texto	Heek	Não	***	Sim
Lincoln Davies	Opções e texto	Lincoln Davies	Sim	*****	Sim
Mobile Monkey	Opções	Mobile Monkey	Sim	**	Não
Peloton	Opções	Peloton	Sim	****	Sim

2.7 Resultados

2.7.1 Alvo

Das onze aplicações testadas, quatro ([App20], [Mic20], [Goo] e [Ama20]) funcionam como um assistente virtual, três ([Amw20], [Mob20] e [Pel20]) funcionam apenas com a hipótese de seleção, ou seja, o *chatbot* oferece várias opções para o utilizador escolher não havendo a possibilidade de uma escrita livre, uma ([Hee20]) apenas funciona por texto, isto é, só pela escrita livre e as restantes três ([Amt20], [Cas20] e [Lin20]) têm a possibilidade de seleção ou de escrita livre.

2.7.2 Disponibilidade

Apenas uma ([Hee20]) aplicação não tem disponibilidade 24/7, ou seja, apenas funciona em horário laboral enquanto todas as outras seja a qualquer altura, é possível iniciar uma conversação.

2.7.3 Apelativo

Quase todas as aplicações testadas são razoavelmente apelativas, com a exceção de duas ([Hee20] e [Mob20]), que visualmente não captam o interesse do utilizador. Esta característica é das mais importantes porque uma aplicação visualmente apelativa, cativa o utilizador e é capaz de simplificar o uso da mesma.

2.7.4 Live Chat

Apenas quatro ([Cas20], [Hee20], [Lin20] e [Pel20]) aplicações têm a opção de *Live Chat*, isto é, a possibilidade de redireccionar para um assistente não virtual em horário laboral. As restantes quando não tem nenhuma resposta para dar ao cliente apenas notifica que não é possível uma resposta, não sendo possível falar com algum assistente.

Capítulo 3

Método Proposto

Neste capítulo é descrito toda a planificação, começando pela abordagem da proposta, nomeadamente o seu desenvolvimento, a sua implementação e, ainda, a apresentação dos resultados de forma conseguir um resultado excelente. Posteriormente apresento um gráfico referente à planificação do trabalho, com todos os dados relevantes.

3.1 Abordagem Proposta

1) Desenvolvimento da proposta de trabalho:
    1.1) Caracterização da Empresa;
    1.2) Descrição do Problema;
    1.3) Estudo do Estado da Arte;
    1.4) Método Proposto;
    1.5) Planificação do Trabalho.
2) Implementação:
    2.1) Análise de Requisitos;
    2.2) Arquitetura do Sistema;
    2.3) Implementação;
    2.4) Testes.
3) Apresentação de Resultados.

3.2 Planificação do Trabalho

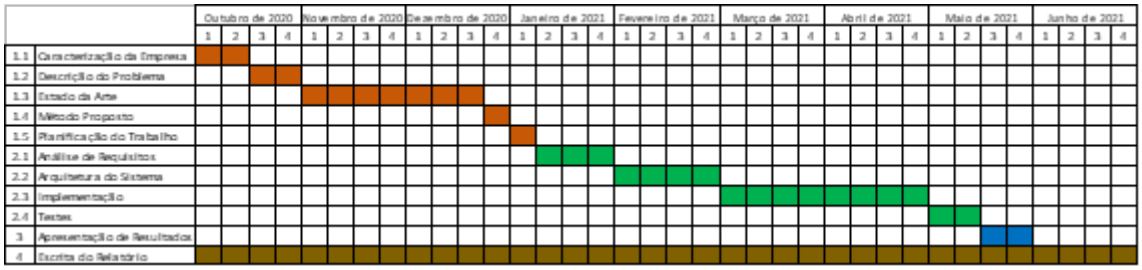

Fig. 3.1: Planificação do Trabalho

Capítulo 4

Implementação e Resultados

4.1 Tecnologias e Ferramentas Utilizadas

4.1.1 MERN

O MERN é uma das variações da *stack* MEAN (MongoDB, ExpressJS, Angular e NodeJS), onde a estrutura de *front-end* tradicional Angular é substituída pelo React. Pode-se dizer que MERN é o nome das quatro tecnologias principais que compõem a *stack* MEAN, ou seja, MongoDB, ExpressJS, React e NodeJS. É, pois, uma *stack* completa que segue o padrão arquitetónico tradicional das três camadas, nomeadamente, a camada de exibição de *front-end* (React), a camada da aplicação (ExpressJS e NodeJS) e ainda, a camada da base de dados (MongoDB).
É de salientar que ExpressJS e NodeJS são as camadas intermediárias.

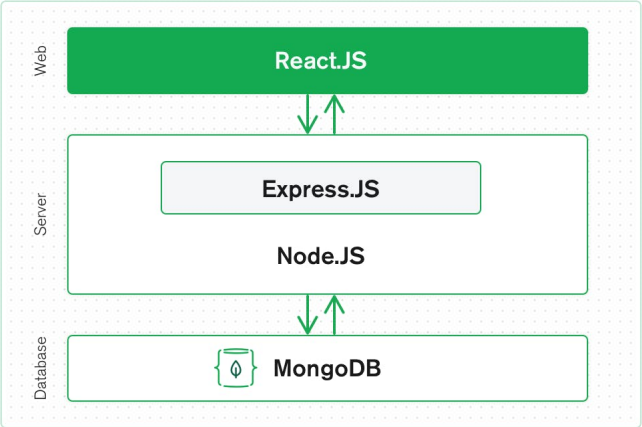

Fig. 4.1: MERN

Vantagem de escolher a stack MERN
O MongoDB é a base de dados de documentos na raiz da *stack* MERN. Ela foi projetada no sentido de armazenar os dados JSON.
Pode-se referir que, o MongoDB funciona muito bem com o NodeJS e facilita o armazenamento e a representação de dados JSON em cada camada da sua aplicação. O ExpressJS e o NodeJS permitem que a *stack* MERN seja mais completa, ou seja, é a estrutura da aplicação do lado do servidor, envolvendo pedidos e respostas HTTP. Além disso, facilita o mapeamento de API's. O ReactJS consiste numa estrutura de *front-end* e tem por finalidade, construir interfaces interativas do utilizador em HTML, para além de comunicar com um servidor (ExpressJS). Assim, pode dizer-se que a combinação do ExpressJS e do

ReactJS é eficaz, já que os dados JSON fluem naturalmente para a frente e para trás.
Para o desenvolvimento de qualquer aplicação, é importante o alinhamento com o mercado e com as tecnologias que o acompanham. O MERN, atualmente, é uma ótima opção para desenvolver uma aplicação *web*.

Arquitetura de 3 camadas
O sistema cliente-servidor é desenvolvido retirando-se a camada de negócio do lado do cliente. O desenvolvimento é o mais demorado visto que tem de dar suporte a uma maior quantidade de plataformas e ambientes diferentes. O retorno vem em forma de respostas mais rápidas nas aquisições, quer em sistemas na *internet* quer em *intranet*, havendo assim, maior controlo no crescimento do sistema.
A arquitetura em três camadas consiste em modelos de programação que permitem distribuir funcionalmente a aplicação por três sistemas distintos, ou seja:

- Componentes que estão em execução nos locais de trabalho;
- Processos que estão em execução nos servidores;
- Coleção discreta de base de dados e possuidores de recursos.

As atualizações e correções de defeitos, podem ser efetuadas sem que haja prejuízo para as restantes camadas. A título de exemplo, podemos referir que as alterações de interface podem ser realizadas sem comprometer as informações obtidas na base de dados e no servidor.

Tipo de camadas

- Camada de apresentação - é chamada GUI (Graphical User Interface), ou apenas Interface. Esta camada interage diretamente com o utilizador e é através dela que se efetuam os pedidos;
- Camada de negócio - é conhecida como lógica empresarial, regra de negócios ou funcionalidade. É nesta camada que ficam as funções e as regras de qualquer negócio. Não existe aqui uma interface para o utilizador e os seus dados são voláteis, isto é, deve ser utilizada a camada de dados para que algum dado consiga manter-se;
- Camada de dados - é composta pela reposição das informações bem como as classes que as manipulam. Pode dizer-se que, esta camada recebe as requisições da camada de negócios; posteriormente os seus métodos executam essas requisições numa base de dados. Caso haja uma alteração na base de dados, apenas seriam alteradas as classes da camada de dados, ficando assim, inalterado o restante da arquitetura.

4.1.1.1 MongoDB

Consiste numa base de dados de documentos livres, *open source* e multi-plataforma. Pode ser classificado como uma base de dados NoSQL. Armazena todos os dados em documentos flexíveis do tipo JSON. Isto significa que os campos podem variar de documento para

documento e a estrutura de dados, pode vir a sofrer alterações ao longo do tempo. O modelo MongoDB é simples, o que o torna fácil de aprender e de usar. Ao mesmo tempo faculta todos os recursos necessário para atender os requisitos mais complexos em qualquer escala.

No MongoDB, a alta disponibilidade, o escalamento horizontal e a distribuição geográfica são integrados e fáceis de utilizar.

As suas características permitem que as aplicações modelem a informação de modo mais natural, visto que os dados podem ser agrupados em hierarquias complexas e continuarem a ser indexados e fáceis de procurar.

4.1.1.2 ExpressJS

É uma *framework* para nodeJS com uma estrutura mínima e flexível, fornecendo um conjunto de recursos, robustos que tem como principal objetivo a construção de servidores *web*. Tendo uma variedade de métodos é possível criar facilmente e rapidamente uma API robusta.

4.1.1.3 React

É uma biblioteca JavaScript *open source*, com o objetivo de criar interfaces gráficas (*front-end*). O React oferece soluções óbvias para certos problemas mais persistentes da programação de *front-end*. Permite assim que, se criem facilmente aplicações dinâmicas e interativas na *web*. É um método rápido, escalável, flexível, poderoso e possui uma comunidade robusta que cresce rapidamente. Consegue fazer imensas funcionalidades de forma rápida e produtiva, devido ao JavaScript.

4.1.1.4 NodeJS

Refere-se a um software *open source*, uma multi-plataforma, que permite a execução de códigos JavaScript fora do *browser*. O *runtime* de JavaScript é constituído pelo node package manager (NPM) e pelo node package executer (NPX). O primeiro tem por função instalar o *package* de forma global ou de forma local enquanto o segundo tem por função executar o código armazenado num *package*. O código de nodejs baseia-se na arquitetura *event-driven*, com capacidade de entrada/saída assíncrona.

Características:
A principal característica e diferença entre as outras tecnologias consiste na execução de eventos em *single-thread*, onde somente uma *thread (Event Loop)* é responsável pela execução do código JavaScript, sem que seja necessário criar uma nova *thread*.

Vantagens:

- Flexibilidade - Possui diversos pacotes e softwares reutilizáveis NPM (Node Packge Manager), dando ao interpretador, um potencial que possa ser utilizado em qualquer situação;
- Eficiência - Não exige muitos recursos computacionais;
- Suporte - Conta com suporte a longo prazo;
- Produtividade - Existem inúmeros pacotes disponíveis gratuitamente que podem ser usados em conjunto.

4.1.2 Ant Design

Ant Design consiste numa biblioteca para React que contém um conjunto de componentes de alta qualidade prontos a serem utilizados. É escrita em linguagem TypeScript. É possível personalizar cada um desses componentes, seja a nível de funcionalidades de cada componente como o seu estilo. Também suporta a internacionalização para diferentes idiomas.
Durante o desenvolvimento foram utilizados os seguintes componentes do Ant Design:

- Button
- Modal
- Table
- Form
- Input
- Result
- Dropdown
- Tag
- Select
- Notification
- Layout
- Row
- Col
- Menu

4.1.3 Webpack

O Webpack consiste num empacotador de módulos estáticos para as aplicações JavaScript modernas. Quando processa a aplicação, o webpack tem a possibilidade de gerar um gráfico que permite mapear cada módulo bem como as suas dependências.

Conceitos principais

- Entry - Consiste no ponto de entrada que indica qual é o módulo que o webpack deve utilizar para começar a construir o gráfico interno de dependência. Quando define o ponto de entrada, o webpack irá encontrar todas as dependências e fazer a sua importação. O ponto de entrada é definido como ./src/index.js. No entanto, é possível definir um arquivo diferente bem como múltiplos pontos de entrada no ficheiro de configuração webpack.config.js;
- Output - A propriedade output define o nome e o local do pacote gerado pelo webpack. O diretório padrão é o ./dist. Quando queremos configurar deve-se definir um objeto output com a propriedade path e filename no ficheiro de configuração do webpack. Tal configuração gerará um diretório dist na raiz do projeto e ainda, o ficheiro main.js dentro do diretório dist;
- Loaders - Os Loaders são importantes para que o webpack consiga entender outros ficheiros que não sejam em JavaScript. São módulos que podem ser instalados separadamente, permitindo que o webpack converta os ficheiros em módulos válidos e os adicione às dependências. Também é de acrescentar que os Loaders são também utilizados para converter JavaScript de uma versão para a outra. Para os Loaders serem incluídos, é criada uma nova secção module no arquivo da configuração, onde podemos incluir uma ou mais regras. Essa regra informa o compilador do webpack que, quando um certo ficheiro é encontrado dentro de uma declaração require ou import, deve-se usar o loader especifico, para assim o transformar e adicionar ao pacote. Dentro do objeto do loader é possível definir quais os ficheiros que serão filtrados (test), definir que módulo deverá ser utilizado (use), tal como os seus próprios plugins.

Exemplo de um ficheiro webpack.config.js:

```
module.exports = {
  entry: "./projects/tf-chatbot-playground/src/index.tsx",
  output: {
    path: path.resolve(__dirname, "dist"),
    publicPath: "/",
    filename: "[name].js",
    chunkFilename: "[id].[hash:8].js",
  },
  module: {
    rules: [
```

```
      {
        test: /\.(woff(2)?|ttf|eot|svg)(\?v=\d+\.\d+\.\d+)?$/,
        use: [
          {
            loader: "file-loader",
            options: {
              name: "[name].[ext]",
              outputPath: "assets/fonts/",
            },
          },
        ],
      },
    ],
  },
};
```

Módulos

O módulo é o nome que se dá a cada ficheiro que exporta uma certa funcionalidade. É funcional dividir uma aplicação em diversos módulos pelos seguintes benefícios:

- Manutenção - é mais fácil alterar uma parte do código sem afetar o sistema inteiro;
- Isolamento - as variáveis e funções dentro de um módulo ficam acessíveis apenas dentro dele, evitando assim, conflitos de variáveis e também o acesso desnecessário de uma parte do código a outra que não esteja relacionada;
- Reusabilidade - um módulo que tenha uma funcionalidade bem definida pode ser reutilizado noutras partes do mesmo sistema ou também noutros sistemas;
- Testabilidade - é mais fácil escrever testes unitários para validar cada funcionalidade.

4.1.4 Redux

O Redux é uma biblioteca JavaScript *open source* que permite gerir o estado da aplicação. Pode dizer-se que o Redux consiste numa solução para o problema da partilha de estados entre os componentes, tornando-o dessa forma, unidirecional. A partir da ilustração, entendemos que o Redux simplifica a evolução dos estados para controlar muitos componentes que precisam de ser atualizados ou modificados.

Para realizar o fluxo, o Redux está dependente de 4 partes, sendo:

- Store - consiste no *container* que tem por função, armazenar e centralizar o estado geral da aplicação. Ela é imutável, isto é, nunca é alterada, apenas evolui. É de salientar que podemos apenas ter uma Store por cada aplicação;

- Actions - são fontes de informação que são enviadas da aplicação para a Store. São simples funções que ativam os Reducers quando são executadas;
- Reducers - tem por finalidade receber e tratar as informações para que, posteriormente, sejam ou não enviadas à Store;
- Conexão dos componentes ao Redux - para poderem ter acesso à Store e conseguirem ver o estado da mesma ou executar certos eventos para a atualizar.

Configuração da Store:

```
const store = createStore(stateReducer, applyMiddleware(thunk));

export { store };


<ReduxProvider store={store}></ReduxProvider>
```

Actions:

```
function userSelectOption(message: Message) {
  return { type: BOT_CONSTANTS.USER_SELECT_OPTION, message };
}

const botActions = {
  userSelectOption
};

export default botActions;
```

O Reducer consiste numa função que fica à escuta de diversas ações e dependendo dessa ação executa lógica para evoluir o estado da Store:

```
const defaultState = {...};

function stateReducer(state = defaultState, action) {
  switch (action.type) {
    case BOT_CONSTANTS.USER_SELECT_OPTION:
      (...)
      return state;
    default:
      return state;
  }
}

export default stateReducer;
```

Conexão com os componentes, em que existe um mapStateToProps (mapear o estado da store para as propriedades do componente) e um mapDispatchToProps (mapear as actions da store para as propriedades do compoente) e depois é feita a conexão com o componente:

```
function mapStateToProps(state) {
  return {state};
}

const mapDispatchToProps = {
  userSelectOption: botActions.userSelectOption
};

const ConnectedShell = connect(mapStateToProps,mapDispatchToProps)(Shell);

export default ConnectedShell;
```

4.2 Modelo e Entidades de Base de Dados

Para o modelo de dados foram utilizadas as seguintes entidades:

- Users (id, username, email, password, name, salt), em que esta entidade tem como função definir os utilizadores do *backoffice*. Cada utilizador é distinguido por um único username e uma *password* em que para uma autenticação segura, cada utilizador tem uma chave única designada por salt;

- Chats (id, status, startDate, clientName, username, endDate, salt), esta entidade tem como objetivo ter a informação de todos os Chats criados pelo Chatbot, tem um estado (status), uma data inicial, ou seja, quando foi feito o pedido da criação do Chat, o nome do cliente (clientName), o username do utilizador do *backoffice* que está a atender um Chat específico, uma data que informa quando é que foi finalizado esse Chat (endDate) e por fim, um salt para ser utilizado na encriptação e desencriptação das mensagens entre os utilizadores;

- Messages (id, message, isBackofficeUser, date, chatId), esta entidade regista todas as mensagens trocadas entre o *backoffice* e o cliente, em que cada mensagem tem um único chatId associado para que haja um *tracking* inteiro de um Chat. Também é guardada uma data, que regista o momento em que a mensagem foi enviada e uma *flag* para haver a distinção entre um cliente e um utilizador do *backoffice*.

Todas estas entidades têm um id único para que não se possam repetir. De seguida podemos então ver o modelo conceptual de dados:

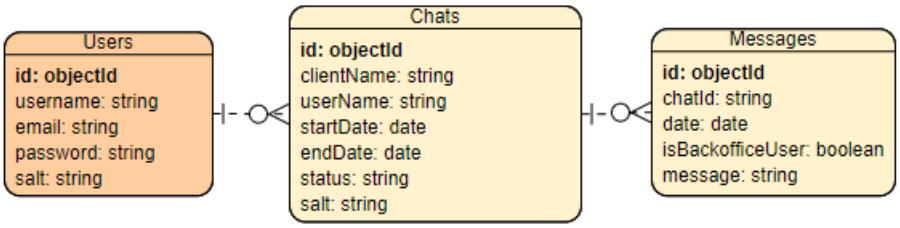

Fig. 4.2: Modelo de Dados

4.3 Segurança da Aplicação

4.3.1 Autenticação e Autorização em aplicações Web

As aplicações *web* são formadas por inúmeros recursos. Na maioria destes recursos existe a necessidade de controlo relativamente ao acesso dos utilizadores, sendo apenas restringidos aqueles que foram previamente identificados e autenticados. Assim, a autenticação representa a maneira como o utilizador prova quem é realmente, enquanto a autorização é usada para constar se o utilizador autenticado possui permissão para utilizar, manipular ou executar o recurso em questão.

A autenticação na *web* é um processo onde é identificado o utilizador que utiliza a aplicação. Isto através de métodos como por exemplo: utilizador/senha, *hash* de autenticação/digest, certificados, integração com outros serviços de identidade, entre outros. Pode-se dizer que, a autenticação básica HTTP é uma forma de autenticar no servidor, onde se adiciona um campo chamado Authorization. O valor desse campo será o *"Basic"*, que consiste na Base64 de uma string.

A autorização na *web* consiste num processo onde se verifica se o utilizador que já foi identificado anteriormente, tem o aceso à função/informação pedida. Para tal pode-se basear em ACLs (Acess Control Lists), perfis, *flags*, entre outros.

4.3.2 JWT

JWT, JSON Web Token, consiste num método RCT 7519 padrão da indústria, que permite autenticar entre duas partes através de um *token* assinado, autenticando uma determinada requisição web. Consiste num código em Base64, armazenando objetos JSON com dados que permitem a autenticação da requisição.

Quando os dados enviados pelo cliente são autenticados no servidor, é criado um *token* JWT assinado com uma *key* interno da API e será enviado o *token* de retorno ao cliente. Assim, tendo um *token* autenticado, o cliente possui o acesso aos recursos da aplicação que anteriormente lhes eram restritos.

Um JWT é composto por três componentes básicos, tais como:

- Header – Consiste na primeira parte do JWT e é composto por dois campos, alg (informa o algoritmo utilizado para criar a *hash* da assinatura) e o tvp (indica que se trata de um token JWT) e consiste no cabeçalho do *token*;
- Payload – É a informação que será enviada. Neste componente encontramos os dados referentes à própria autenticação;
- Signature - Consiste na assinatura única de cada *token* e é gerada a partir de um algoritmo de criptografia. Baseia-se no *header*, no *payload* e no *secret* estabelecido pela aplicação. Consiste pois, numa palavra chave que é utilizada para codificar e não deve ser partilhada por ninguém.

4.3.3 Autenticação e Autorização no Chatbot e no Livechat

Para gerar um *token* JWT é necessária a biblioteca jsonwebtoken, em que o username será o utilizador a ser autorizado, expiresIn será a duração deste *token* e o process.env.TOKENSECRET será a *secret key* para assinar o *token*:

```
const jwt = require('jsonwebtoken');

function generateAccessToken(username) {
  return jwt.sign(username, process.env.TOKENSECRET, { expiresIn: '1800s' });
}
```

Para validar um *token* e proteger as API's, é necessário voltar utilizar a biblioteca jsonwebtoken, em que com o *token* recebido é validado pela *secret key*, process.env.TOKENSECRET:

```
const jwt = require('jsonwebtoken');

export function authenticateToken(req, res, next) {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (token == null) return res.sendStatus(401);

  jwt.verify(token, process.env.TOKENSECRET, (err, user) => {
    if (err) return next(new ErrorHandler(HTTPCODE.UNAUTHORIZED));

    req.user = user;

    next();
  });
}
```

Devido a estes dois métodos, é possível gerar um *token* quando é feito um *login* na aplicação e de seguida fazer a validação desse mesmo *token* ao utilizar as API's protegidas.

```
router.post('/api', authenticateToken, async function (req, res, next) {...}
```

4.3.4 Encriptação

A encriptação consiste num processo que transforma a informação de maneira que alguém não autorizado não consiga lê-la. Para conseguir necessita de uma chave própria, *hash*, que lhe permite descodificar essa mesma informação, revertendo-a para a sua forma original. Existem inúmeros métodos para encriptar e desencriptar a informação. Devido à encriptação, as empresas conseguem proteger os seus dados e torná-los seguros.

Pode-se referir que, o objetivo da encriptação é garantir a privacidade, protegendo os dados digitais durante o seu envio. Os algoritmos da criptografia conseguem fornecer a segurança, permitindo a verificação da origem da mensagem e a sua integridade.

Neste projeto, a encriptação é utilizada quando um utilizador faz *login*, ou seja, a password é enviada em *hash* para o servidor e o servidor verifica se para o utilizador encontrado a password é igual à password recebida descodificada com o *hash* do utilizador encontrado. A outra utilização da encriptação é quando é enviada alguma mensagem, a mensagem é codificada com o *hash* do próprio chat e descodificada da mesma forma.

4.4 Chatbot e Live Chat

4.4.1 Estrutura do Projeto

Este projeto encontra-se dividido por 4 projetos. Sendo eles, o Chatbot (1), o Live Chat (3), o Live Chat Server (4) e um Playground (2). O Playground serve para testar todos os outros 2 projetos juntamente com o Live Chat Server a correr, ou seja, uma aplicação em que está integrada com o componente do Chatbot e o componente do Live Chat.

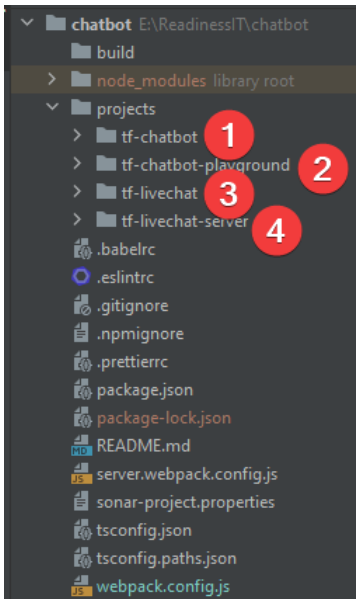

Fig. 4.3: Estrutura do Projeto

Integração do Chatbot no Playground, em que como input temos toda a configuração opcional e customizável (configuration) e a linguagem, que é atualizada conforme é mudada no Playground. Por fim, temos como output o navigate que é quando o utilizador pede para o Chatbot mostrar a página resultante da resposta automática do Chatbot:

```
<Chatbot
    configuration={botConfiguration}
    navigate={onBotNavigate}
    language={language}/>
```

Integração do Livechat no Playground, em que também existe dois inputs, um para a configuração customizável e outro para a linguagem que está a ser utilizada no Playground:

```
<Livechat
    configuration={configuration}
    language={language}/>
```

Para além desta integração dos componentes no Playground, também é possível fazer o build tanto do Chatbot como do Livechat e do Livechat Server para serem integrados noutros projetos sem que se tenha o código fonte. Para isso é necessário fazer o build e o publish para um repositório NPM, neste caso é feito o *upload* do *package* para o Nexus da ReadinessIT.
O *build* dos *packages* é feito de uma forma diferente do Playground, ou seja, na configuração do webpack.config.js o mode é prodution (modo produção), leva os devidos loaders e o output será um ficheiro tf-livechat.js ou tf-chatbot.js. Para fazer o build são executados os seguintes comandos por ordem:

- rimraf ./build mkdir build - Irá apagar a pasta do build e criar uma pasta nova;
- webpack –config webpack.config.js - Faz o build do package;
- npm publish - É feito a publicação da biblioteca para o Nexus.

Caso seja a primeira vez que se está a fazer npm publish ou npm install, será necessário fazer o npm set registry para o URL onde se encontram as bibliotecas, neste caso é o Nexus e após fazer o set registry é necessário fazer o npm login para autenticar um utilizador e que seja permitido aceder ao Nexus.
Para ser instalado uma biblioteca e integrar em qualquer projeto, é só preciso adicionar ao package.json o nome da *library* às dependências e fazer npm install. Depois é só importar no componente em que vai ser utilizada esta biblioteca.

package.json:

```
"dependencies": {
    "tf-livechat":"^1.0.0",
    "tf-chatbot":"^1.0.0"
}
```

component.tsx:

```
import tf-livechat from 'tf-livechat';
import tf-chatbot from 'tf-chatbot';

<tf-chatbot
    configuration={botConfiguration}
    navigate={onBotNavigate}
    language={language}/>

<tf-livechat
    configuration={configuration}
    language={language}/>
```

4.4.2 Estrutura do Chatbot

```
Shell
.. ChatHeader
.... MinusOutlined
.. ChatContent
.... ChatBody
...... RestartAnimation
...... TypingAnimation
...... ChatBubble
........ ChatTag
.......... Tag
.......... Button
........ LiveChatChoice
.......... Tag
.. ChatFooter
.... ChatInput
.... ChatInputSubOptions
...... MenuOutlined
...... Input
.. Button
.... WechatOutlined
```

4.4.3 Estrutura do Livechat

```
Shell
.. LiveChatSideNav
.... Link
.... Menu
...... MenuItem
.. LiveChatContent
.... LiveChatMessenger
...... LiveChatBubble
...... LiveChatMessengerActions
........ LiveChatEndModal
.......... Modal
............ Result
.............. Button
........ Button
...... Input
........ SendOutlined
.... LiveChatDashboard
...... Input
...... Select
........ Option
...... Button
...... LiveChatSessions
```

```
.... LiveChatLogin
...... Input
...... Button
```

4.4.4 Principais componentes

4.4.4.1 TypingAnimation

Cada vez que existe uma mensagem do lado do bot, em progresso, é mostrada uma animação de que dá a entender que alguém está a escrever, para dar uma percepção ao utilizador que tem de aguardar pela resposta,

```
{props.botMessageInProgress > 0 ? <TypingAnimation /> : null}
```

```
import "./TypingAnimation.less";
import React from "react";

const TypingAnimation = (props) => {
  return (
    <div className="typing-container">
      {props.message ? <span className="mb-1">{props.message}</span> : null}
      <div className="typing-animation">
        <span />
        <span />
        <span />
      </div>
    </div>
  );
};

export default TypingAnimation;
```

4.4.4.2 ChatBubble

O ChatBubble mostra a mensagem, data, hora e quem enviou. Este componente tem dois comportamentos, o comportamento de quando é o Chatbot/Livechat a enviar uma mensagem ou quando é o cliente. Ou seja, quando é o cliente a mensagem fica do lado direito com uma cor e quando é o Chatbot/Livechat fica do lado esquerdo com uma cor diferente.

```
<div
  className={`bubble container mb-1 ${
    isMessageFromBot ? "" : "bubble-direction-reverse"
  }`}
>
  <Row className={`${isMessageFromBot ? "" : "justify-content-flex-end"}`}>
    <span className={`bubble-name `}>{message.name}</span>
```

```
  </Row>
  <div className={`bubble ${isMessageFromBot ? "bot" : "user"}`}>
    {groupConfiguration ? (
      <div className="mb-1">
        <span className="bubble-message mb-1">
          {getGroupMessageValue(
            groupConfiguration.message,
            groupConfiguration.translateKey
          )}
        </span>
      </div>
    ) : null}
  </div>
</div>
```

4.4.4.3 Input do Chatbot

O Input do Chatbot tem uma particularidade de mostrar um input e dois botões, ou seja, onde o cliente escreve a mensagem e depois um botão para enviar e outro botão para mostrar várias opções, essas opções permitem ir para o *website* da ReadinessIT, reiniciar o Chatbot ou fechar a sessão caso esteja em Livechat.

```
<div>
  <Col>
    <Input
      disabled={props.botStopped || waitingForReply}
      value={message}
      onPressEnter={onClick}
      bordered={false}
      onChange={onChangeMessage}
      className="footer-input"
      placeholder={t("place-holders.write-response")}
      addonBefore={
        BOT_CONFIGURATION.SHOW_INPUT_SUB_OPTIONS ? (
          <ChatInputSubOptions />
        ) : null
      }
      addonAfter={
        <SendOutlined
          style={{ border: 0 }}
          onClick={onClick}
          disabled={props.botStopped || waitingForReply}
        />
      }
    />
  </Col>
</div>
```

4.4.4.4 Dashboard do Livechat

O Dashboard contém um Input para filtrar pelo número do chat e um Select que mostra os filtros possíveis para encontrar uma listagem de chats abertos, fechados e/ou em progresso. Logo em baixo destes filtros, é representada a listagem dos resultados.

```
<span className="input-title">{t("inputs.conversation-id")}</span>
<Input
  bordered={false}
  onChange={handleChatIdFilter}
  className="live-chat-input"
  value={selectedChatId}
/>

<span className="input-title">{t("inputs.conversation-state")}</span>;
<Select
  mode="multiple"
  className="status-filters"
  onChange={handleStatusFilter}
  value={selectedStatus}
  optionLabelProp="label"
>
  {statusDropdownOptions.map((option) => {
    return (
      <Option value={option.action} label={t(option.translationKey)}>
        <span>{t(option.translationKey)}</span>
      </Option>
    );
  })}
</Select>;

<LiveChatSessions />
```

4.4.4.5 Sessões do Livechat

As sessões são mostradas em forma da uma tabela que contém várias colunas, das quais são: número do chat, nome do cliente, data, estado e ações.

```
<Table
  className="table-config"
  columns={columns}
  dataSource={sessions}
  bordered={false}
  size="middle"
  rowClassName="data-row"
  scroll={{ y: 500 }}
/>;
```

4.4.4.6 Notificações

Sempre que haja algum tipo de erro ou quando é feito algo com sucesso, é mostrada uma notificação no ecrã.

```
import { notification } from 'antd';

export function openNotification(message, type?, placement?, description?) {
  switch (type) {
    case 'info':
      notification.info({
        message,
        placement: placement || 'topRight',
        description,
      });
      break;
    case 'warning':
      notification.warning({
        message,
        placement: placement || 'topRight',
        description,
      });
      break;
    case 'error':
      notification.error({
        message,
        placement: placement || 'topRight',
        description,
      });
      break;
    case 'success':
      notification.success({
        message,
        placement: placement || 'topRight',
        description,
      });
      break;
  }
}
```

4.4.5 Funções auxiliares

4.4.5.1 ObjectPathExists

Quando queremos retirar um campo de um objeto em que esse objeto está dentro de outro objeto e temos de percorrer esses dois objetos para chegar a esse campo, existe a possibilidade de um objeto intermédio não existir e depois não ser possível retirar esse mesmo campo pois não existe e a aplicação pode ficar visualmente estranha devido a ter ocorrido

um erro na consola. Então esta função percorre objetos sem que ocorra algum tipo de erro. Por exemplo, objetoA.objetoB.objetoC.field1, em que se o objectoB não existe já vai aparecer um erro na consola. Então usando esta função simplesmente dizemos que queremos ir ao objectoA percorrer o caminho objectoB.objectoC.field1 e que queremos o valor desse field1, objectPathExists(objetoA, 'objetoB.objetoC.field1', true).

```
export function objectPathExists(
  o: object,
  path: string,
  getValue?: boolean
): boolean | any {
  if (!o || !Object.keys(o).length) return false;
  const keys = path.split(".");
  const loop = () => keys.every((k) => ((o = o[k]), isDefined(o)));
  const passFail = loop();
  return !getValue ? passFail : passFail && o;
}
```

4.4.5.2 Cast

Esta função tem como objetivo transformar um objeto em JSON numa classe, ou seja, na aplicação Chatbot existem duas classes, uma classe de configuração e uma classe que representa as mensagens e quando guardamos toda essa informação na storage, guardamos em forma de JSON e quando é necessário carregar essa informação na storage para a aplicação é importante fazer um cast de JSON para a classe em sí, para que depois seja possivel utilizar as funções e get's presentes na classe.

```
export function cast(rawObj, constructor) {
  const obj = new constructor();
  for (let i in rawObj) obj[i] = rawObj[i];
  return obj;
}
```

4.4.5.3 BreakObjectRef

Quando queremos modificar um objeto temporariamente sem que seja modificado o objeto principal utilizamos esta função que simplesmente faz uma cópia do objeto e que depois seja possível modificar esse objeto que é retornado da função sem que seja modificado o objeto principal.

```
export function breakObjectRef(object: object): any {
  return JSON.parse(JSON.stringify(object));
}
```

4.4.5.4 FormatDate e FormatTwoDigits

A função formatDate serve para transformar uma data em uma string que seja entendível. Já a formatTwoDigits tem a função de transformar números com um digito em dois digitos, ou seja, 2 passa para 02 mas 10 fica 10.

```
export function formatDate(date: Date) {
  if (!(date instanceof Date)) date = new Date(date);
  const h = formatTwoDigits(date.getHours());
  const m = formatTwoDigits(date.getMinutes());
  const s = formatTwoDigits(date.getSeconds());
  return h + ':' + m + ':' + s;
}

function formatTwoDigits(digit: number) {
  return digit < 10 ? '0' + digit : digit;
}
```

4.4.6 Serviços Implementados

4.4.6.1 Serviço Axios

Devido à necessidade de utilizar o JWT para autenticar e autorizar o utilizador, é necessário, em cada pedido HTTP, seja adicionado um header de Authorization e para isso acontecer foi necessário criar uma instância de um serviço já existente, chamado Axios.

Este serviço, para quando existe um token, intercepta um pedido HTTP e adiciona esse token ao header de Authorization:

```
const originalAxios = require('axios');

function axios() {
  function assignHeadersInterceptors() {
    const headers = {};
    if (store) {
      const storeState = store.getState();

      const token = objectPathExists(
        storeState,
        'authenticationReducer.token',
        true
      );

      if (token) Object.assign(headers, { Authorization: `Bearer ${token}` });
    }
    return headers;
  }
```

```
  return originalAxios.create({
    baseURL: serverUrl(),
    headers: assignHeadersInterceptors(),
  });
}

export default axios;
```

4.4.6.2 Serviço de Traduções

A nível de traduções, houve a necessidade de também criar um serviço para cada projeto, ou seja, um serviço para o Chatbot e um serviço para o Livechat. Esta necessidade deve-se a quando se juntam os dois projetos num só, termos duas instâncias de forma a separar as traduções do Chatbot das traduções do Livechat. Isto não era possível, sem a criação destes dois serviços.

A titulo de exemplo, temos a instância do Livechat :

```
function loadResources() {
  const resourcesToLoad = {};

  Object.values(LANGUAGES).forEach((value) => {
    const translationFile = require(`../../translations/${value}.json`);

    if (translationFile)
      resourcesToLoad[value] = { translation: translationFile };
  });

  return resourcesToLoad;
}

const liveChatInstance = i18next.createInstance();

liveChatInstance.use(initReactI18next).init({
  resources: loadResources(),
  lng: LIVE_CHAT_CONFIGURATIONS.DEFAULT_LANGUAGE,
  fallbackLng: 'en-US',
  interpolation: {
    escapeValue: false,
  }
});

export default liveChatInstance;
```

4.4.6.3 Serviço de encriptação

Para a encriptação foi também criado um serviço só para isso, que não contém nada mais que duas funções, uma para encriptar e outra para desencriptar. Ambas recebem uma string e um salt, em que através da biblioteca crypto-es, é possível encriptar essa mesma string com uma chave secreta (salt).

```
import cryptoES from 'crypto-es';

function encryptString (string: string, salt: string) {
  return cryptoES.AES.encrypt(string, salt).toString();
}

function decryptString (string: string, salt: string) {
  return cryptoES.AES.decrypt(string, salt).toString(cryptoES.enc.Utf8);
}

const encryptService = {encryptString, decryptString};

export default encryptService;
```

4.4.6.4 Serviço para comunicação com as API's

Para a comunicação entre o *front-end* e o *back-end* foi necessário, mais uma vez, a criação de um serviço para cada projeto. Esses serviços contêm funções para comunicar com as API's.

O Chatbot utiliza apenas uma API, /livechat/createChat, para ser criado um novo chat com um estado em aberto para depois este ser atendido por alguém responsável do lado do Livechat.

Quanto ao Livechat, já utiliza mais que uma API, das quais são:

- /authentication/signin - Login com um *username* e *password* em que é devolvido um *token* JWT;
- /authentication/refresh-token - Quando um *token* JWT está prestes a ser expirado e ainda existe um utilizador autenticado, é gerado um novo *token*;
- /livechat/getSessions - Para obter todos os chats existentes;
- /livechat/endSession - Para terminar uma sessão, ou seja, para encerrar um chat;
- /livechat/startSession - Para iniciar uma sessão, ou seja, para iniciar a atenção a um chat;
- /livechat/chatMessages - Para buscar todas as mensagens dado um número de chat único.

Na execução destas API's é utilizado o serviço Axios para enviar automaticamente o header de autorização, a título de exemplo:

```
import axios from '../http/axios';

function endSession(chatId: string) {
  return axios()
    .patch(loadRoute('endSession'), { chatId })
    .then((res) => res)
    .catch(() => null);
}
```

Listagem de API's

POST - /authentication/signin

```
Request:
{
  "username": "string",
  "password": "string"
}

Response:
{
  "status": "string",
  "data": {
    "username": "string",
    "clientId": "string",
    "name": "string",
    "token": "string"
  }
}
```

POST - /authentication/refresh-token

```
Request:
{
  "username": "string",
}

Response:
{
  "status": "string",
  "data": {
    "token": "string"
  }
}
```

POST - /livechat/getSessions

```
Request:
{
  "maxResults": "string",
  "chatId": "string",
  "status": "string"
}

Response:
{
  "status": "string",
  "data": [
    {
      "clientName": "string",
      "startDate": "date",
      "endDate": "date",
      "status": "string",
      "chatId": "string",
      "username": "string",
      "salt": "string"
    }
  ]
}
```

PATCH - /livechat/endSession

```
Request:
{
  "chatId": "string",
}

Response:
{
  "status": "string"
}
```

PATCH - /livechat/startSession

```
Request:
{
  "username": "string",
  "chatId": "string"
}

Response:
```

```
{
  "status": "string"
}
```

GET - /livechat/chatMessages/:chatId

```
Params:
- chatId: "string"

Response:
{
  "status": "string",
  "data": [
    {
      "chatId": "string",
      "message": "string",
      "isBackofficeUser": "boolean",
      "data": "date"
    }
  ]
}
```

POST - /livechat/createChat

```
Request:
{
  "clientName": "string",
}

Response:
{
  "status": "string",
  "data": {
      "clientName": "string",
      "startDate": "date",
      "status": "string",
      "chatId": "string",
      "salt": "string"
    }
}
```

4.4.7 Classes Implementadas

4.4.7.1 Class de Configuração

```
export class Configuration {
```

```
private _configuration: BotConfiguration;

constructor(configuration: BotConfiguration) {...}

public updateSelf(newConfiguration: BotConfiguration) {...}

get language() {...}

get preloadedParams() {...}

get defaultName() {...}

get translations() {...}

get welcomeMessage() {...}

get configuration() {...}

get maxAttempts() {...}

get liveChat() {...}

get routes() {...}

set configuration(configuration: BotConfiguration) {...}

get groupConfiguration() {...}

get initialQuestions() {...}

get sortedInitialQuestions() {...}

sortByPriority(messages: { match: number; message: Message }[]) {...}

assignIdsToMessages(configuration: BotConfiguration) {...}

get firstInitialQuestion() {...}

get messages() {...}

hasMoreAttempts(currentAttempts: number) {...}

findMessageByKeywords(keywords: string = '', exact?: boolean): Message |
    Message[] {...}

getOptionGroupMessageByGroup(group: string) {...}

getOptionsByMessages(messages: Message[] = []) {...}
```

```
  findRouteByRouteKey(message: Message) {...}
}
```

4.4.7.2 Class de Mensagens

```
export class Message {
  private _message: BotMessage;

  constructor(message: BotMessage) {...}

  public updateSelf(message: BotMessage) {...}

  get message() {...}

  set message(message: BotMessage) {...}

  get order() {...}

  get messageValue() {...}

  get paramsToUse() {...}

  get messageType() {...}

  get messageId() {...}

  get keywords() {...}

  get routeKey() {...}

  get regex() {...}

  set regex(regex: RegExp | string) {...}

  get paramToSave() {...}

  get group() {...}

  get regexFailMessage() {...}

  get replyFromOption() {...}

  get searchForExactKeyword() {...}

  get translationKey() {...}
```

```
  get regexTranslationKey() {...}

  isEqual(message: Message) {...}

  clone() {...}

  isValidMessage(value: string) {...}

  get failedValidationMessage() {...}
}
```

4.4.8 Configurações

4.4.8.1 Chatbot Messages

Para o Chatbot funcionar corretamente é necessária alguma configuração, em que é necessário um conjunto de entradas:

- language - Linguagem pré-definida para ser utilizada;
- name - Nome do Chatbot;
- messages - Listagem de todas as mensagens automáticas do Chatbot. As mensagens podem ser categorizadas em 4 tipos, OPTION, INITIALQUESTION, RESPONSE e WELCOME. O tipo OPTION vai agrupar uma série de mensagens que tenham o mesmo grupo e vai mostrar como várias opções de escolha. O tipo INITIALQUESTION são as mensagens iniciais que tem por objetivo perguntar ao cliente uma série de perguntas, por exemplo, qual o seu nome, o seu email ou o seu contacto. O tipo RESPONSE simplesmente faz correspondência ao que o cliente escreveu e responde normalmente, sem a possibilidade de ser possível selecionar alguma opção ou navegar para alguma página. Por último, o tipo WELCOME que irá ter apenas uma mensagem para dar as boas vindas ao cliente. Todas as mensagens têm um valor e este poderá ser traduzido ou não, depende se tiver uma chave de tradução. No caso das mensagens INITIALQUESTION, poderá ser guardada a resposta do cliente numa variável dinâmica declarada na configuração (paramToSave) em que depois estes parâmetros podem ser utilizados, por exemplo, nas mensagens de tipo WELCOME em que esta mensagem tem um parâmetro, também dinâmico, paramsToUse, em que faz correspondência. Todas as mensagens do tipo INITIALQUESTION têm uma ordem para serem executadas conforme o input do cliente. Também é possível validar a resposta do cliente, ou seja, se for um email podemos inserir um regex para validar o input e caso não corresponda poderá ser enviada uma mensagem, traduzida ou não. Para as mensagens do tipo OPTION, é possível definir uma rota para depois navegar para a página que o cliente escolheu;
- routes - Todas as rotas para que quando o Chatbot sugere visitar uma página e caso o utilizador escolha essa sugestão, o Chatbot redirecionar automaticamente para essa

página (rota);

- preloadParams - Caso existam certos parâmetros já vindos, por exemplo, de um login, e que tenhamos o nome do cliente, email ou número de contacto, sejam carregados automaticamente para o Chatbot e para que o Chatbot possa usar toda essa informação nas suas respostas automáticas;
- groupConfiguration - Quando existe mais que uma opção para a mesma iteração com o utilizador, são agrupadas as mensagens e pode ser possível ou não mostrar uma mensagem geral juntamente com todas as opções disponíveis;
- maxAttepnts - Número de máximas tentativas de erro para o Chatbot peça para redirecionar até ao Livechat, se tiver disponível, ou pedir para encerrar a conversação atual;
- liveChat - Boleano para saber se o Livechat está disponível ou não;
- translations - Para o Chatbot carregar todas as traduções customizáveis para depois serem apresentadas corretamente. Estas traduções são depois adicionadas às traduções já existentes por defeito, sendo possível alterar depois as mensagens de erro que estão definidas por defeito no Chatbot.

Interfaces necessárias para a configuração do Chatbot:

```
export enum LANGUAGES {
  ENGLISH = 'en-US',
  PORTUGUESE = 'pt-PT',
}

export enum MESSAGE_TYPE {
  OPTION,
  INITIAL_QUESTION,
  RESPONSE,
  WELCOME,
}

export interface BotConfiguration {
  language: LANGUAGES;
  name: string;
  messages: BotMessage[];
  routes?: BotRoute[];
  preloadParams?: MessageParamSaved[];
  groupConfiguration?: BotGroupConfiguration[];
  maxAttempts: number;
  liveChat: boolean;
  translations?: BotTranslation[];
}
```

```
export interface BotMessage {
  messageType: MESSAGE_TYPE;
  messageValue: string;
  messageTranslationKey?: string;
  keywords?: string[];
  routeKey?: string;
  regex?: RegExp | string;
  paramsToUse?: string[];
  paramToSave?: string;
  order?: number;
  group?: string;
  regexFailMessage?: string;
  regexFailMessageTranslationKey?: string;
  replyFromOption?: boolean;
  searchForExactKeyword?: string;
}

export interface BotRoute {
  routeKey: string;
  routeValue: string;
}

export interface MessageParamSaved {
  paramName: string;
  paramValue: string;
}

export interface BotGroupConfiguration {
  key?: string;
  message?: string;
  translateKey?: string;
}

export interface BotTranslation {
  language: string;
  json: string;
}
```

Configuração exemplo do Chatbot:

```
export const configuration: BotConfiguration = {
  language: LANGUAGES.PORTUGUESE,
  name: 'ReadinessIT Bot',
  maxAttempts: 10,
  liveChat: true,
  messages: [
    {
```

```
    messageType: MESSAGE_TYPE.INITIAL_QUESTION,
    order: 1,
    messageValue: 'Can you tell me your name?',
    regexFailMessage:
      'That name is not valid. Can you repeat your name please?',
    paramToSave: 'name',
    regex: /^[a-zA-Z ,.'-]+$/i,
  },
  {
    messageType: MESSAGE_TYPE.INITIAL_QUESTION,
    order: 2,
    messageValue: 'Welcome %name%! What I can do for you?',
    paramsToUse: ['name'],
  },
  {
    messageType: MESSAGE_TYPE.RESPONSE,
    messageValue: 'We can list our products, smartphones, simcards',
    messageTranslationKey: 'response.help',
    keywords: ['help', 'me'],
  },
  {
    messageType: MESSAGE_TYPE.OPTION,
    messageValue: 'Samsung Note 20',
    group: 'smartphones',
    routeKey: 'samsungN20',
    keywords: ['smartphones', 'smart', 'phones'],
  },
  {
    messageType: MESSAGE_TYPE.OPTION,
    messageValue: 'Samsung Note 20+',
    group: 'smartphones',
    routeKey: 'samsungN20plus',
    keywords: ['smartphones', 'smart', 'phones'],
  },
  {
    messageType: MESSAGE_TYPE.WELCOME,
    messageValue: "Hello, I'm ReadinessIT Bot",
  },
],
routes: [
{
    routeKey: 'products',
    routeValue: 'products',
  },
  {
    routeKey: 'samsungS20',
    routeValue: 'products/1',
  },
```

```
    ],
  groupConfiguration: [
  {
      key: 'smartphones',
      message: 'Here are the results that you searched for:',
      translateKey: 'group.show-results',
    },
  ],
  preloadParams: [
    {
      paramValue: '+351912123412',
      paramName: 'phone',
    },
  ],
  translations: [
    {
      language: 'pt-PT',
      json: JSON.stringify(pt),
    }
  ],
};
```

4.4.8.2 Configurações default do Chatbot e do Livechat

```
const CACHE_PERSIST_TIME = 15; // Tempo para que a cache seja válida, se passar
    esse tempo, a cache é limpa.
const DEFAULT_BOT_TITLE = 'ReadinessIT'; //Nome do bot por defeito.
const DEFAULT_LANGUAGE = LANGUAGES.ENGLISH; //Linguagem por defeito.
const DEFAULT_ERROR_MESSAGE = 'Something went wrong'; //Mensagem de erro por
    defeito.
```

4.4.9 Processos automáticos

Por vezes, existem sessões que ficam abertas por inatividade ou porque o cliente simplesmente desistiu e desligou o Chatbot, então existe a necessidade de ter um processo automático para encerrar automaticamente esses chats que ficaram em aberto. Este processo pega em todos os registos que existem na entidade Chats e que estejam em progresso ou em aberto e que a data seja com pelo menos um dia de diferença e atualiza para um estado de fechado e que a data final seja a data em que se atualizou esse estado, ou seja, a data de quando o processo é executado. Este processo é executado hora a hora.

```
import processes from './core/processes';

processes.closeOpenSessions();

function closeOpenSessions() {
```

```
  setInterval(function() {
    console.debug('Running close open sessions process at: ' + new
        Date().toLocaleString());
    const Chat = require('../model/chats');
    const todayDate = new Date();
    todayDate.setDate(todayDate.getDate() - 1);

    const query = {
      status: { $in: [SESSION_STATUS.IN_PROGRESS, SESSION_STATUS.OPEN] },
      startDate: { $lt: todayDate.toISOString() },
    };

    Chat.updateMany(
      query,
      {
        status: SESSION_STATUS.CLOSED,
        endDate: new Date(),
      }).then((res) => console.debug(res)).catch(err => console.debug(err));
  }, 60 * 60 * 1000);
}

const processes = {
  closeOpenSessions,
};

export default processes;
```

4.4.10 Comunicação entre o Chatbot e o Livechat

Para haver uma comunicação entre o Cliente e a pessoa que está a atender esse cliente, existia duas hipóteses, uma que era fazer uma *poll* de x em x segundos sempre à procura de uma nova mensagem, tanto por parte do Cliente tanto por parte da pessoa que está a atender ou então implementar algo que transmitisse dados em tempo real, para isso foi utilizado o mecanismo de sockets. Este foi implementado através da biblioteca SocketIO.

4.4.10.1 Servidor

Do lado do servidor, é criado um servidor socket em que é possível fazer várias transações, começando pela initSession, que vai registar um utilizador a um certo chatId (identificador único de um Chat), para que seja possível utilizar as diferentes transações possíveis e uma delas, a mais importante, é quando o utilizador envia uma mensagem e esta mesma transação emite essa mensagem para quem está a atender. Ainda assim, no meio destas transações, ainda é guardado na base de dados essa mensagem para futuras consultas.

```
const server = http.createServer(app);
const io = socketIo(server, {
```

```
  cors: {
    origin: '*',
  },
});

require('./socket/socket.io')(io);

io.on('connect', (socket) => {
    socket.on('initSession', ({ chatId }) => {
      socket.join(chatId);
    });

      socket.on('USER_SEND_MESSAGE', ({ chatId, message, name }) => {
      transactions.sendMessage(chatId, message, false).then((res) => {
        if (res) io.to(chatId).emit('RECEIVED_USER_MESSAGE', { message, name });
      });
    });
});
```

4.4.10.2 Cliente

Do lado do cliente, é necessário fazer o initSession, enviado o chatId para que seja possível o registo do utilizador nessa sessão e depois de ter um socket conectado, podemos-o utilizar para enviar mensagens, essas mensagens vão sempre encriptadas com o hash (salt) registada no próprio Chat.

```
function createSocket(username, chatId) {
  const serverUrl = environment.serverUrl || '/';

  const socket = io(serverUrl);

  socket.emit('initSession', { username, chatId }, (error) => {
    if (error) {
      alert(error);
    }
  });

  return socket;
}

function userSendMessage(socket, chatId, message, name, salt) {
  socket.emit(IO_CONSTANTS.USER_SEND_MESSAGE, {
    chatId,
    message: encryptService.encryptString(message, salt),
    name,
  });
}
```

4.5 Resultados

Na figura 4.4 mostra a inicialização do Chatbot, ou seja, a mensagem inicial configurada como WELCOME e de seguida as mensagens configuradas como INITIALQUESTION. As mensagens têm uma ordem especifica, configurável, e tal como mostra na figura 4.5 é possível validar a resposta que o cliente deu.

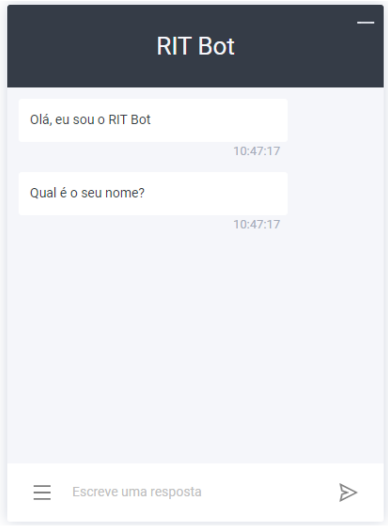

Fig. 4.4: Introdução e primeira pergunta inicial

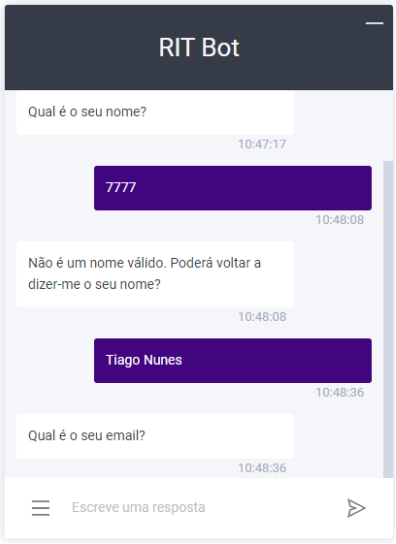

Fig. 4.5: Regex configurado numa mensagem

Na figura 4.6 é mostrada uma mensagem final a dar as boas vindas ao cliente, tratando-o(a) pelo nome que inseriu na pergunta inicial. Já na figura 4.7 mostra um resultado daquilo que o cliente pediu.

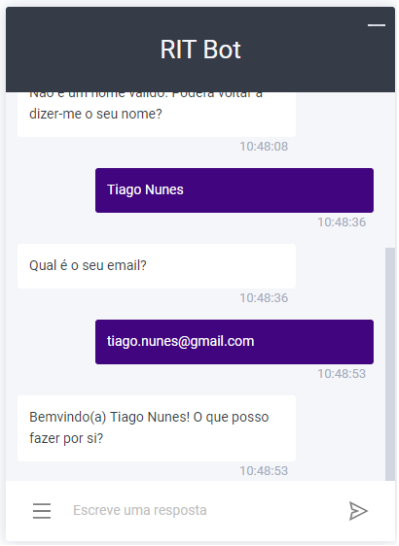

Fig. 4.6: Mensagem de bem vindo

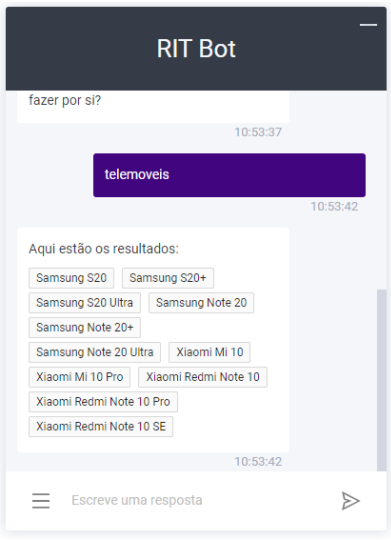

Fig. 4.7: Resultados de uma pesquisa

Na figura 4.8, o cliente selecionou um equipamento da lista e a aplicação redirecionou a página para o equipamento selecionado, já na figura 4.9, como não existe configuração da rota para aquela opção, é enviada uma mensagem de erro.

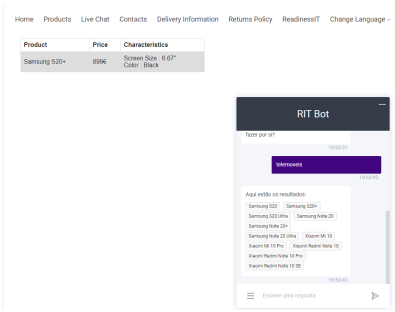

Fig. 4.8: Resultado da navegação para o produto desejado

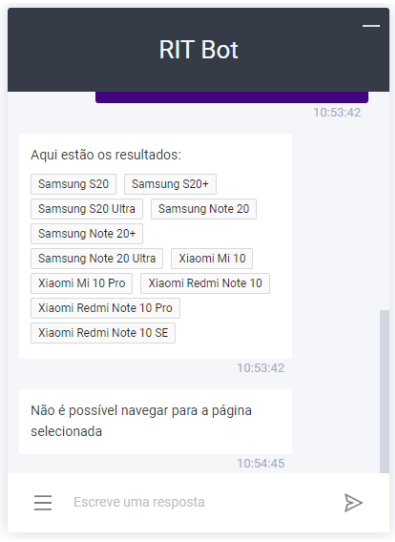

Fig. 4.9: Navegação para o produto desejado sem sucesso

Na figura 4.10 o cliente queria saber mais sobre informações de entrega e então o Chatbot permitiu que o cliente fosse redirecionado para a página onde está toda essa informação.

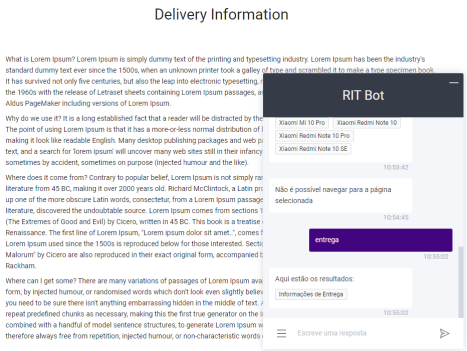

Fig. 4.10: Resultado da navegação para a informação desejada

Na figura 4.11 é mostrada apenas uma animação que aparece sempre que o Chatbot tiver a processar a mensagem de resposta ao cliente. Na figura 4.12, quando o Chatbot já não consegue responder às necessidades do cliente, pede então que seja redirecionado para um suporte live, em que irá ter algúem do outro lado para o ajudar.

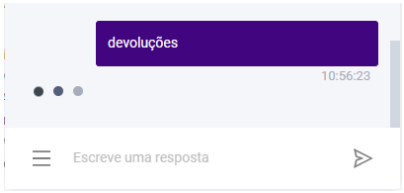

Fig. 4.11: Animação de quando o Chatbot está a responder

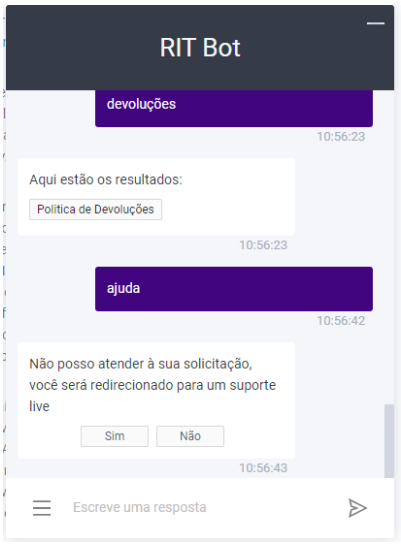

Fig. 4.12: Solicitação para ser atendido por um suporte live

Na figura 4.13 mostra um submenu que existe no Chatbot para reiniciar a conversação e na figura 4.14 é mostrada uma animação de todo esse processo.

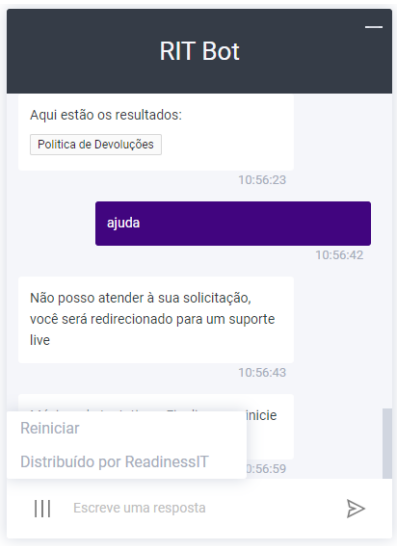

Fig. 4.13: Submenu para reiniciar o Chatbot

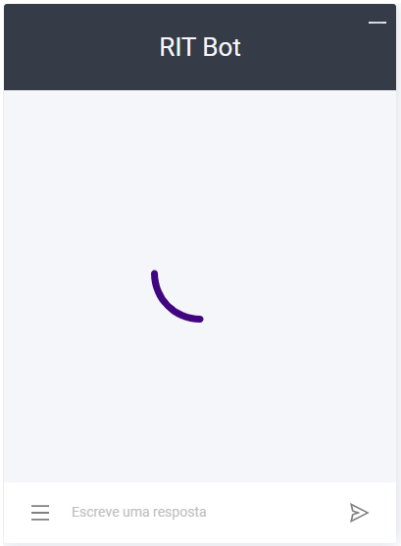

Fig. 4.14: Animação de quando o Chatbot está a reiniciar

Quando o live suport é recusado aparece uma mensagem a pedir para finalizar e que as tentativas máximas foram excedidas, figura 4.15 e se for aceite aparece uma mensagem a avisar que o cliente está em lista de espera para ser atendido, figura 4.16.

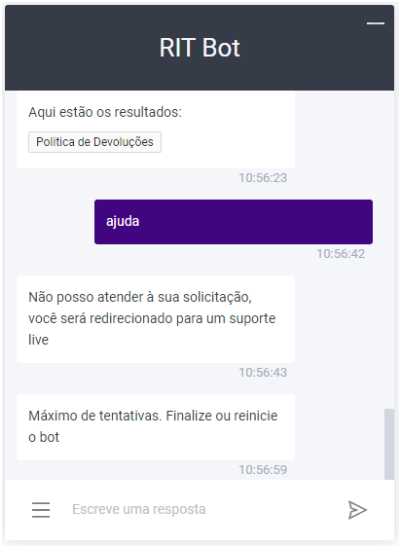

Fig. 4.15: Suporte live recusado

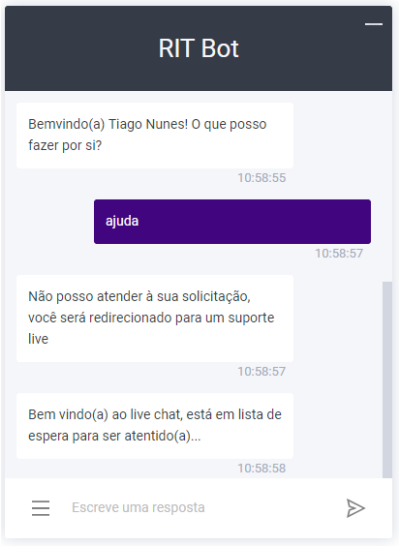

Fig. 4.16: Suporte live aceite

Na figura 4.17 mostra o ecrã de inicio de sessão do live chat em que aqui é que é feita a autenticação de um utilizador.

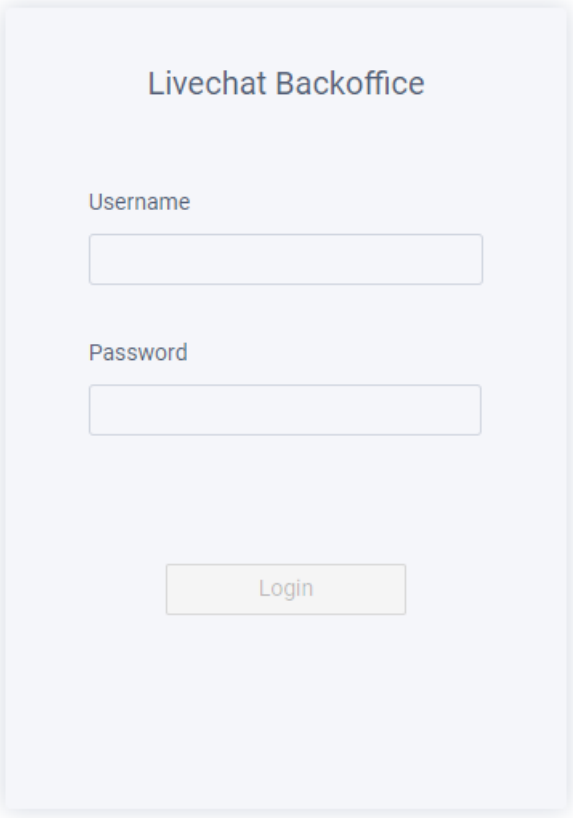

Fig. 4.17: Login Livechat

Caso seja um login com sucesso mostra uma notificação de sucesso, figura 4.18, e caso seja sem sucesso mostra uma notificação de erro, figura 4.19.

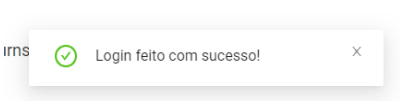

Fig. 4.18: Notificação de Login com sucesso

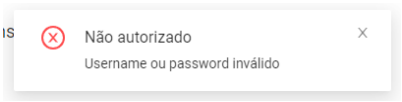

Fig. 4.19: Notificação de Login sem sucesso

Na figura 4.20 é representado o dashboard do Livechat em que:

- 1 - Visualização das mensagens desse chat;
- 2 - Fechar ou cancelar um chat;
- 3 - Começar a atenção de um chat;
- 4 - Número do chat;
- 5 - Nome do cliente;
- 6 - Data em que foi feito o pedido de live suport;
- 7 - Estado do chat em questão;
- 8 - Filtros do estado de conversação;
- 9 - Filtro para o número de chat;
- 10 - Botão para reiniciar todos os filtros inseridos;
- 11 - Botão para procurar chats com ou sem filtros;
- 12 - Botão para fazer logout.

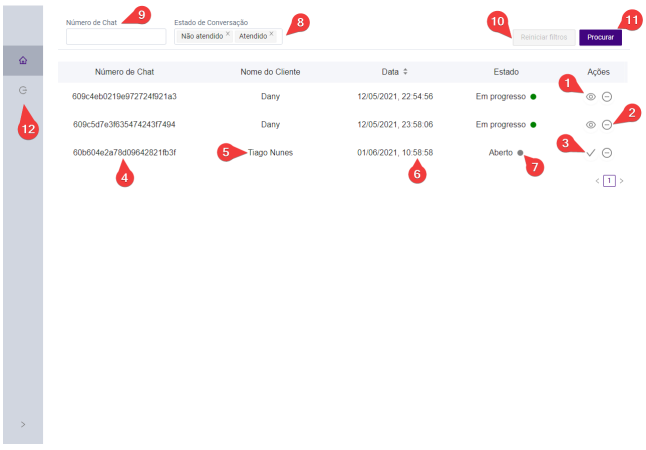

Fig. 4.20: Dashboard Livechat

Na figura 4.21 mostra uma notificação quando é começada uma atenção e redireciona para uma nova página fora do dashboard, na figura 4.22 é mostrado um feedback de que foi iniciada a atenção em que é disponibilizado o nome da pessoa que está a atender.

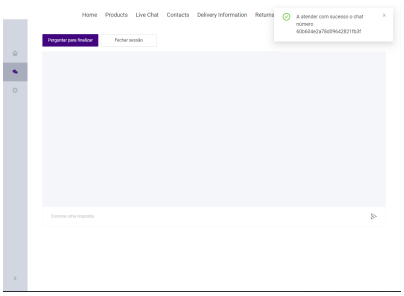

Fig. 4.21: Começo da atenção Livechat

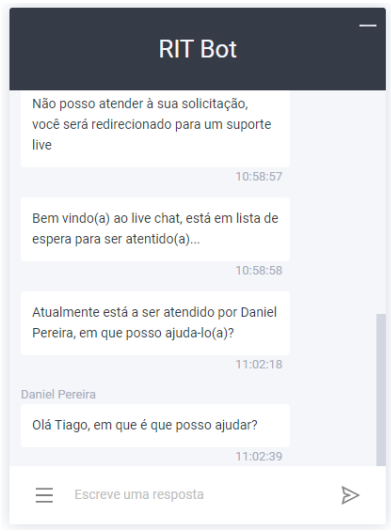

Fig. 4.22: Feedback do começo da atenção Livechat no Chatbot

Na figura 4.23 e na figura 4.24 mostra a conversação entre um componente e o outro. Na figura 4.23 é a perspetiva no Livechat e na figura 4.24 é a perspetiva no Chatbot

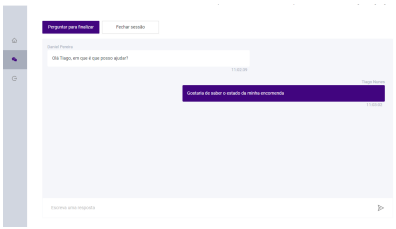

Fig. 4.23: Livechat: chat entre Livechat e Chatbot

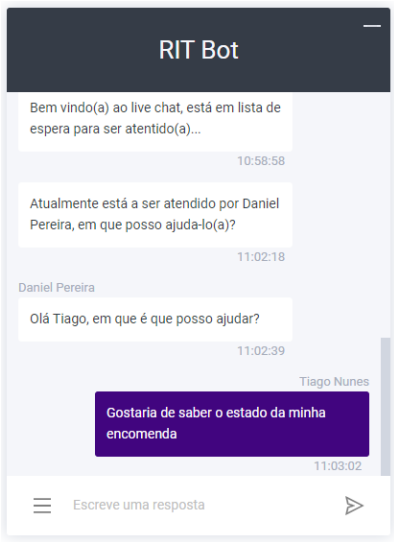

Fig. 4.24: Chatbot: chat entre Livechat e Chatbot

Na figura 4.25 o utilizador do Livechat pergunta ao cliente se tem mais alguma questão e se não tiver, é pedido para finalizar o chat.

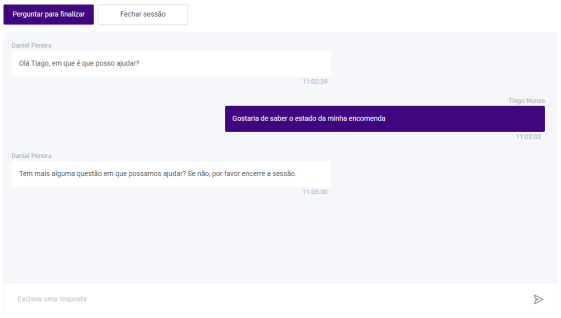

Fig. 4.25: Utilizador Livechat pede para finalizar

É possível, sempre que o cliente queira, finalizar o chat no submenu do Chatbot, tal como mostra a figura 4.26. Após a finalização do chat, é mostrada uma mensagem em que essa sessão foi encerrada, figura 4.27.

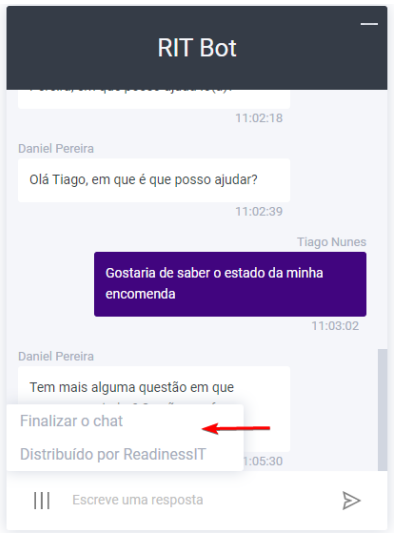

Fig. 4.26: Submenu do Chatbot para finalizar o chat

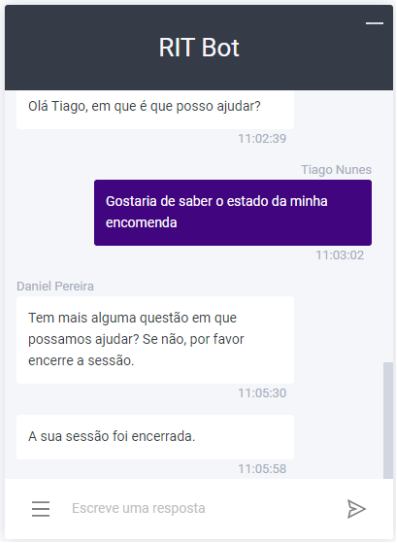

Fig. 4.27: Sessão encerrada no Chatbot

Quando o cliente finaliza um chat, no lado do Livechat aparece uma modal com uma notificação de que o utilizador encerrou o live chat, figura 4.28.

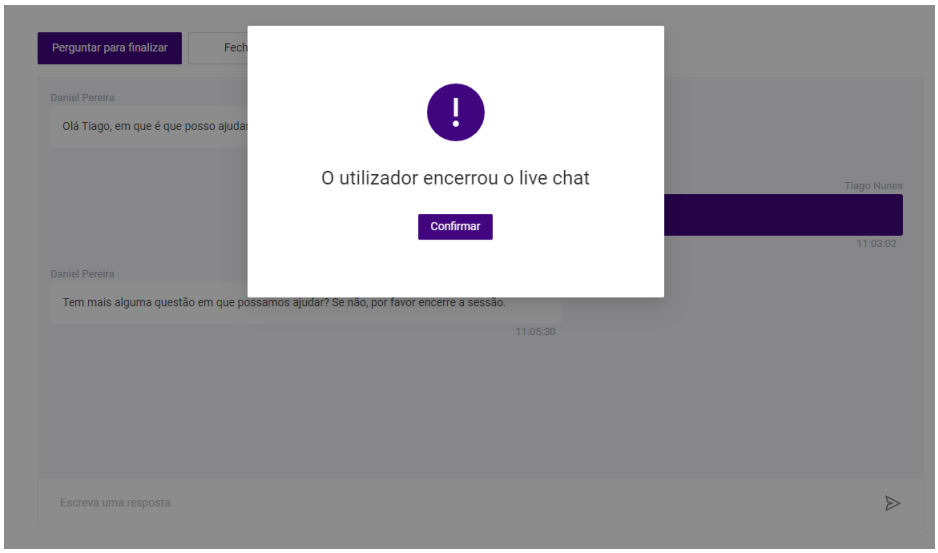

Fig. 4.28: Notificação no Livechat de que o utilizador encerrou o chat

Todos os chats ficam guardados na base de dados, sendo possível ver nos resultados do dashboard com os devidos filtros, como mostra a figura 4.29.

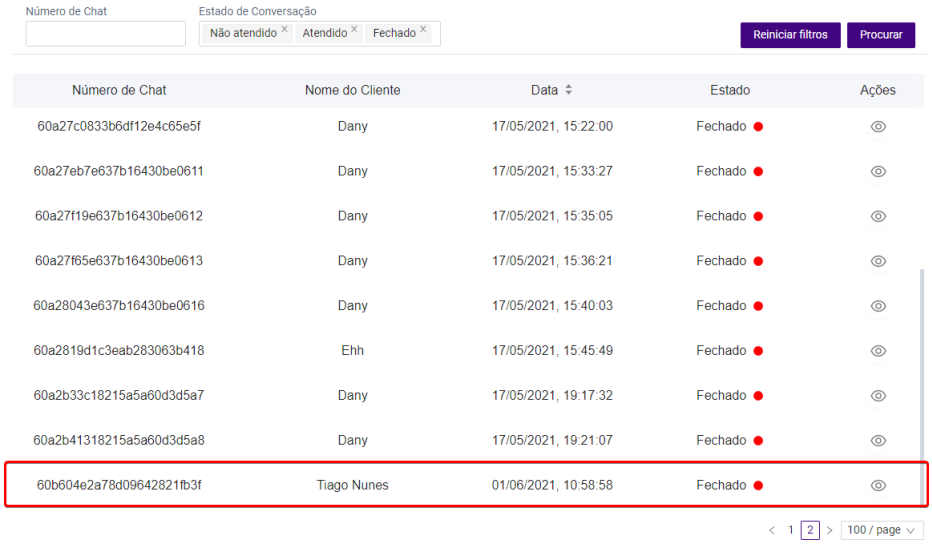

Fig. 4.29: Sessão fechada nos resultados

Na figura 4.30 podemos ver que com o número do chat conseguimos ver todo o histórico de mensagens entre o cliente e a pessoa que o atendeu.

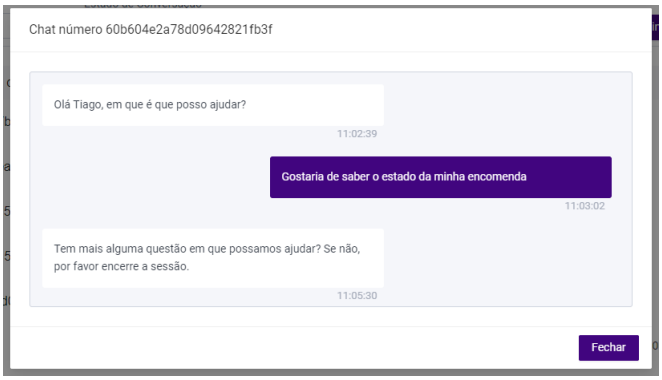

Fig. 4.30: Visualização do historial de mensagens de um Chat

Na figura 4.31 mostra o caso de um utilizador já esteja a atender algum cliente mas não se tenha apercebido (1) e queira começar a atender outro cliente diferente (2) é mostrada uma notificação de informação de que deve terminar a atenção atual primeiro (3).

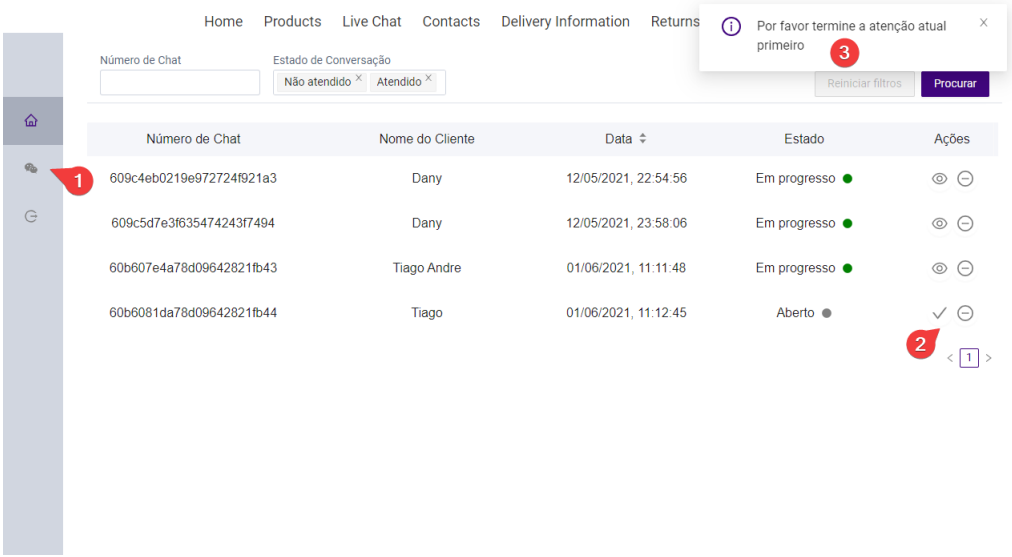

Fig. 4.31: Iniciar uma sessão com uma sessão em curso

Quando é o utilizador do Livechat a encerrar a sessão, figura 4.32, é mostrada uma mensagem no Chatbot de que a sessão foi fechada por esse utilizador, figura 4.33.

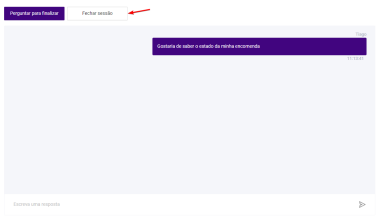

Fig. 4.32: Fechar sessão do lado do Livechat

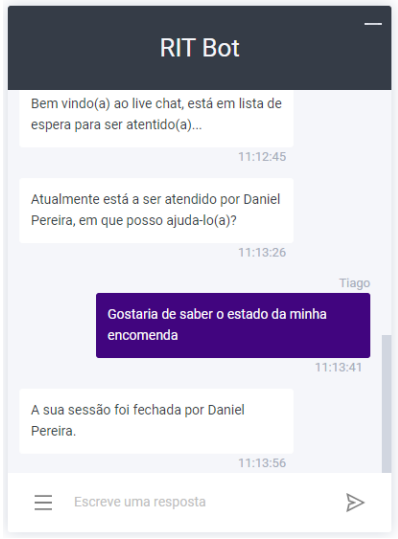

Fig. 4.33: Sessão encerrada no Chatbot

Existem diferentes filtros dos quais são: Não atendido, atendido e fechado. Mostra na figura 4.34.

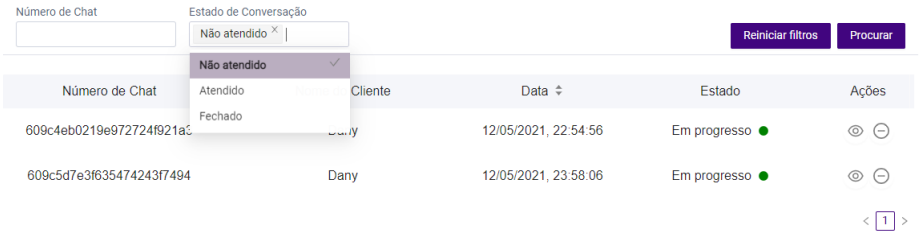

Fig. 4.34: Filtro do estado de conversação

Na figura 4.35 mostra o resultado com filtros aplicados, número de chat e estado de conversação.

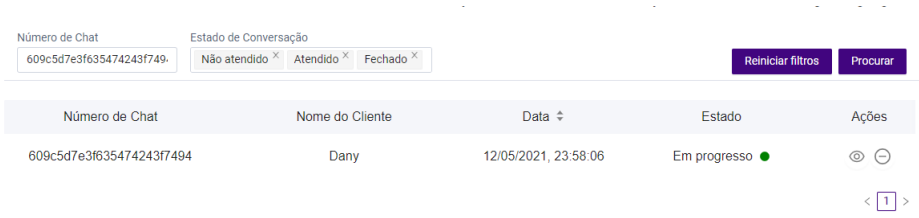

Fig. 4.35: Resultado de uma pesquisa com os filtros

Na figura 4.36 mostra resultados de chats em aberto de há vários dias atrás, antes de correr o processo automático implementado.

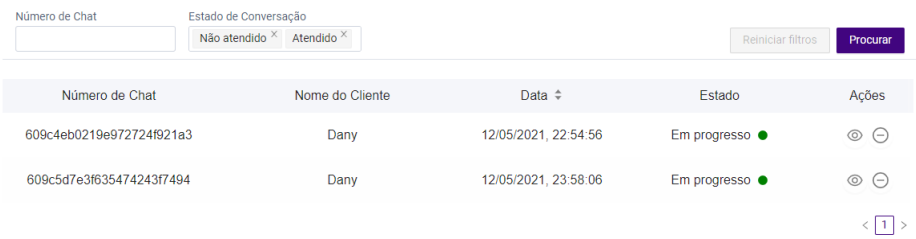

Fig. 4.36: Resultados antigos antes de correr o processo automático

Já na figura 4.37, é mostrado que foi corrido um processo a uma certa data em que foram modificados 2 registos e que foi ok.

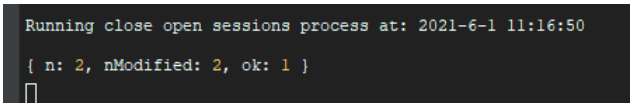

Fig. 4.37: Resultado do feedback do processo automático na consola

De seguida, como mostra na figura 4.38, esses chats que apareciam antes do processo automático, já não aparecem.

Fig. 4.38: Resultados antigos depois de correr o processo automático

Na figura 4.39 mostra uma lista de chats abertos mais recentes (no próprio dia).

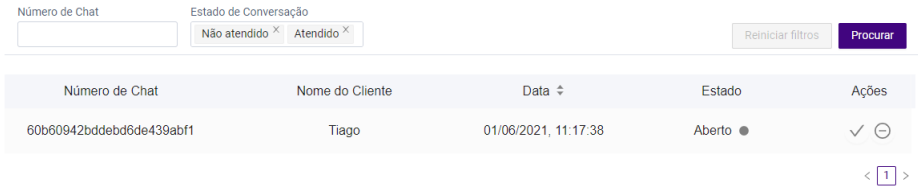

Fig. 4.39: Resultados recentes antes de correr o processo automático

Em que o processo automático não modificou nenhum registo, pois não existem registos com mais de um dia, figura 4.40.

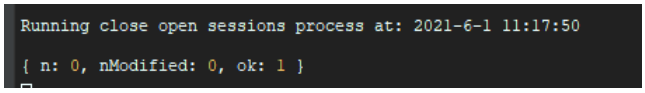

Fig. 4.40: Resultado do feedback do processo automático na consola

Capítulo 5

Conclusão

5.1 Conclusões Principais

Com o desenvolvimento deste projeto aprimorei o meu conhecimento de como criar um projeto de raiz e aprendi várias tecnologias, que para mim, foram novas e que irão influência o meu futuro pois poderá abrir novas portas para novos projetos. Na realização deste projeto, foram encontrados alguns problemas e dificuldades, mas foi possível ultrapassar sem problemas e sem grande dificuldade com a ajuda de pessoas mais seniores na ReadinessIT e com a ajuda do motor de busca, o Google. Estes problemas e dificuldades serviram para adquirir novos conhecimentos pois se eles não aparecessem apenas seria aplicado o conhecimento que já tinha adquirido em todo o meu percurso escolar e profissional.

5.2 Trabalho Futuro

Primeiramente, para trabalho futuro, fica a sugestão de melhorar o Chatbot e implementar um bocado de inteligência artificial e implementar um Backoffice para que todas as configurações sejam facilmente alteradas ou mesmo adicionadas novas configurações e que sejam guardadas numa base de dados em vez de se passar como input quando se chama a biblioteca no HTML. Posteriormente, fica uma vontade de poder lançar atualizações na aplicação com novas funcionalidades e principalmente corrigir bugs que vão surgindo durante a sua utilização.

Bibliografia

[Ama20] Amazon. Alexa [online]. 2020. Available from: https://alexa.amazon.com/ [cited 8 Dezembro 2020]. 14

[Amt20] Amtrak. Amtrak [online]. 2020. Available from: https://www.amtrak.com/home [cited 8 Dezembro 2020]. 14

[Amw20] Amwell. Amwell [online]. 2020. Available from: https://business.amwell.com/ [cited 8 Dezembro 2020]. 14

[App20] Apple. Siri [online]. 2020. Available from: https://www.apple.com/siri/ [cited 8 Dezembro 2020]. 14

[AW20] Ritu Agarwal and Mani Wadhwa. Review of state-of-the-art design techniques for chatbots. *SN Computer Science*, 1, 07 2020. 8, 9, 10

[Cah17] Jack Cahn. Chatbot: Architecture, design, development, volume = 1. 04 2017. 8, 10

[Cas20] Casper. Casper [online]. 2020. Available from: https://casper.com/home-v1/ [cited 8 Dezembro 2020]. 14, 15

[CCM+17] Gillian Cameron, David Cameron, Gavin Megaw, Raymond Bond, Maurice Mulvenna, Siobhan O'Neill, Cherie Armour, and Michael McTear. Towards a chatbot for digital counselling. January 2017. 31st International BCS Human Computer Interaction Conference: Digital Make Believe, HCI 2017 ; Conference date: 03-07-2017 Through 06-07-2017. 10

[Goo] [online]Available from: https://assistant.google.com/,author={{Google}},title={GoogleAssistant},year=2020,lastchecked={8Dezembro2020},note={}. 14

[Hee20] Heek. Heek [online]. 2020. Available from: https://www.heek.com/ [cited 8 Dezembro 2020]. 14, 15

[Lin20] Lincoln Davies. Lincoln davies [online]. 2020. Available from: https://www.lincolndavies.com/ [cited 8 Dezembro 2020]. 14, 15

[Mic20] Microsoft. Cortana [online]. 2020. Available from: https://www.microsoft.com/pt-pt [cited 8 Dezembro 2020]. 14

[Mob20] MobileMonkey. Mobilemonkey [online]. 2020. Available from: https://mobilemonkey.com/ [cited 8 Dezembro 2020]. 14

[Pel20] Peloton. Peloton [online]. 2020. Available from: https://www.onepeloton.com/ [cited 8 Dezembro 2020]. 14, 15

[Rah12] J. Rahman. Implementation of alice chatbot as domain specific knowledge bot for brac u (faq bot). 2012. 10

[SA02] BA Shawar and E Atwell. *A comparison between Alice and Elizabeth chatbot systems*. University of Leeds, School of Computing research report 2002.19, December 2002. Shawar, BA and Atwell, E (c) 2002, University of Leeds. Reproduced with permission from the copyright holders. Available from: `http://eprints.whiterose.ac.uk/81930/`. 10

[SB16] Reshmi Sankar and Kannan Balakrishnan. Implementation of an inquisitive chatbot for database supported knowledge bases. *Sādhanā*, 41:1–6, 10 2016. 10