INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Serviço de Helpdesk automático

DIOGO GUILHERME MARQUES DOS SANTOS
Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Serviço de Helpdesk automático

Diogo Guilherme Marques dos Santos

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

Orientador: António Constantino Lopes Martins

Co-Orientador: Paulo Rogério Soares Proença

Porto, Outubro 2020

Resumo

Cada vez mais as organizações percebem que é importante investir no apoio e suporte aos clientes. Numa perspetiva tecnológica esta necessidade é ainda mais premente, atualmente, são utilizadas plataformas ou aplicações tecnológicas para a maior parte das ações do quotidiano.

Muitas destas organizações adotam o uso de redes sociais - ou outros canais - para oferecer suporte aos seus clientes. No entanto, em alguns casos estas podem não ser as melhores opções, porque não se adaptam ao seu negócio ou são pouco profissionais.

Assim sendo, o objetivo desta dissertação é estudar e acompanhar o desenvolvimento de um serviço de *help desk*. Uma plataforma que visa melhorar a interação entre o cliente e o departamento de suporte de uma empresa, em situações de ajuda ou apoio a um cliente.

Para este sistema são desenvolvidas duas plataformas baseadas em arquiteturas de micro serviços. Uma das aplicações sustenta a relação com o cliente e a outra implementa a lógica associada ao funcionamento do departamento de suporte da empresa. Este sistema adapta-se a qualquer organização independentemente do seu negócio ou clientes.

Concluiu-se que a plataforma desenvolvida cumpre com os requisitos analisados e estabelecidos. De acordo com inquéritos a vários utilizadores esta poderia ser uma melhoria para os processos de suporte entre uma organização e os seus clientes.

Palavras-chave: help desk, serviço, suporte a clientes

Abstract

The importance of customer support is more and more a priority and some organizations are starting to invest more in this subject. From a technologic perspective, this necessity is even greater since most of the everyday actions are performed with a software application or platform.

Many of these organizations use social networks or different channels to offer support to their customers, however, these may not be the ideal options, because they may not adapt to the company´s business or they may be unprofessional.

Due to this, this dissertation intends to study and develop a help desk software. A platform that aims to improve the interaction between a customer and the support department of a company.

For this system, two microservices-based applications are developed, one of these provides the communication with the customer while the other implements the logic associated with the company´s support department. This system should be able to be adopted by any platform or organization regardless of their business or customer base.

With the system developed, it was concluded this fulfills the analyzed and established requirements. According to inquiries made to several users, this could be an improvement to the customer support process between an organization and its customers.

Keywords: help desk, service, customer support

Agradecimentos

Primeiramente, quero deixar claro o meu profundo agradecimento aos meus pais e à minha namorada, que sempre foram um suporte, uma fonte de motivação e que me fizeram querer alcançar todos os meus objetivos – quer a nível académico/profissional como a nível pessoal. Aproveito, também, para agradecer aos meus restantes familiares e amigos chegados, pelo apoio, encorajamento e acompanhamento neste percurso.

Por fim, mas não menos importante, agradeço ao Instituto Superior de Engenharia, do Politécnico do Porto e aos seus docentes, pelos conhecimentos que me foram transmitidos e pelo suporte que me foi dado, quer no mestrado como anteriormente na Licenciatura. Entre os docentes, destaco o Dr. Paulo Proença e o Dr. Constantino Martins, por terem orientado o desenvolvimento desta dissertação e pela disponibilidade e ajuda fornecida. Foi um longo percurso, e deste mesmo percurso levo valores e lições, que sei que vão caminhar comigo tanto a nível profissional como a nível pessoal.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

B2B	*Business to Business*
B2C	*Business to Consumer*
CI	*Continuous Integration*
CD	*Continuous Delivery*
HTTP	*Hyper Text Transfer Protocol*
HTTPS	*Hyper Text Transfer Protocol Secure*
SOA	Service Oricented Arquitectures
FURPS	*Funtionality Usability Reliability Performance Suportability*
SDK	*Software Development Kit*
UC	*Use Case*
CRUD	*Create Read Update Delete*
STS	*Security Token Service*
DAL	*Data Access Layer*
BLL	*Business Logic Layer*
DB	*Database*
DTO	*Data Transfer Object*

1 Introdução

Este capítulo tem por objetivo introduzir o projeto a ser desenvolvido, começando pela sua contextualização. Posteriormente, é descrito o problema a resolver, os principais objetivos a serem alcançados e a abordagem adotada para a elaboração de uma solução ideal. Por último, é elencada a estrutura deste documento para tornar o processo de leitura mais simples e organizado.

1.1 Contexto

Com o rápido crescimento da informatização, grande parte da sociedade começou a utilizar plataformas informáticas para diversas funções do quotidiano, desde pequenas leituras de notícias, a processos num ambiente profissional. Este crescimento também aumenta a competitividade entre estas plataformas, surgindo a necessidade de agradar cada vez mais aos utilizadores e investir cada vez mais na usabilidade e simplicidade das plataformas.

No entanto, por muito simples que uma plataforma possa ser, existem sempre obstáculos ou questões que podem surgir aos utilizadores. Quando isto acontece, é necessário um processo de suporte a estes utilizadores para que os obstáculos possam ser ultrapassados e seja mantida uma boa impressão da plataforma. Para efetuar este suporte, as empresas possuem um departamento específico, que pode ser denominado de *help desk*.

Este departamento é responsável por efetuar o primeiro contacto com o utilizador, perceber o problema em causa e ajudar na sua resolução ou, caso seja necessário, delegar o trabalho a outro departamento mais competente. Das várias formas de estabelecer contacto com um departamento de suporte, destacam-se as chamadas telefónicas, e-mails e, mais recentemente, a utilização de funcionalidades embutidas nas plataformas ou redes sociais. “It is specially focused on end user functionality, and, thus, is responsible for quick resolution of immediate needs, incidents and technical issues of end users.” (Brown, 2015).

As diversas plataformas tecnológicas podem ser separadas em 2 categorias, *Business to Business (B2B)* ou *Business to Consumer (B2C).* Uma plataforma B2B é uma plataforma desenvolvida para outra empresa, adaptando-se á sua lógica de negócio e necessidades, enquanto que uma plataforma B2C é dirigida ao público geral. Estas categorias influenciam o tipo de questões que os utilizadores da plataforma podem ter.

Para este documento, um utilizador é um cliente da empresa que necessita de suporte, e um colaborador representa o funcionário desta empresa que irá prestar o suporte ao utilizador.

1.2 Problema

O processo de suporte aos utilizadores é bastante importante para o sucesso de qualquer plataforma, pois reduz a sensação de frustração do utilizador e melhora a usabilidade da plataforma. Assim sendo, é importante que este processo seja o mais eficiente e prestável possível, para melhorar a comunicação e a gestão das questões colocadas internamente, pelo utilizador.

Para este efeito, surge a necessidade do desenvolvimento de um sistema que ajude a melhorar este processo. Este sistema deve proporcionar uma comunicação entre os utilizadores de uma plataforma e os colaboradores responsáveis pelo seu apoio. Internamente este sistema deve ser capaz de gerir e fornecer toda a informação necessária para aumentar a eficiência da resolução dos tickets (pedidos dos utilizadores).

De acordo com (Rongala, 2015), um sistema de *help desk* é indispensável para qualquer empresa com uma plataforma informática, sendo estes os principais benefícios:

- Satisfação dos Clientes;
- Melhoria na qualidade dos produtos/serviços;
- Melhoria na produtividade;
- Retenção de custos.

1.3 Objetivo

Para solucionar o problema será necessário o desenvolvimento de um sistema de *help desk* para comunicação entre os utilizadores de uma plataforma e a empresa que a desenvolveu.

Sendo assim, o objetivo deste projeto é o desenvolvimento do sistema acima referido. Este deve cumprir todos os requisitos principais de um sistema de *help desk* - os clientes da organização devem ser capazes de colocar as suas questões e problemas no sistema e acompanhar o seu desenvolvimento, assim como comunicar eficientemente com algum membro da organização. Do lado da organização, os tickets devem ser geridos da forma mais automática possível, para melhorar a eficiência do processo de suporte.

Para cumprir estes objetivos, devem ser consideradas as seguintes funcionalidades:

1- Criação de tickets e gestão de tickets previamente criados: Os utilizadores devem conseguir criar um ticket (que representa uma questão) na plataforma, posteriormente, este ticket deve poder ser acedido pelo próprio. Os colaboradores devem ser capazes de gerir o ticket e efetuar as alterações necessárias;
2- Visualização de alterações: O Colaborador deve ser capaz de visualizar todas as alterações efetuadas a um ticket.
3- Chat entre o Colaborador e o Utilizador: O Sistema deve ser capaz de gerar uma interface para comunicação entre os intervenientes, através de um chat em tempo real;
4- Notificações a utilizadores aquando de alterações no ticket: Quando são realizadas alterações a um ticket, os intervenientes devem receber notificações acerca das mesmas. Estas notificações devem ser cuidadosamente analisadas para selecionar as mais relevantes;
5- Distribuição automática de tickets: O Sistema deve ser capaz de, automaticamente, atribuir um ticket criado a um colaborador;
6- Possibilidade de reatribuição de tickets: Um colaborador deve ser capaz de reatribuir o ticket a outro colaborador ou departamento;
7- Visão de análise administrativa: O sistema deve também oferecer uma visão administrativa, onde se poderá observar os dados estatísticos destes processos, permitindo a um administrador a gestão de todos os recursos e tickets, assim como uma perspetiva da eficiência e eficácia dos seus recursos;
8- Integrabilidade em qualquer negócio ou plataforma: Este sistema deve ser facilmente integrado em qualquer plataforma web assim como qualquer tipo de negócio.

1.4 Abordagem

Para resolver este problema, existem algumas soluções que podem ser implementadas. Visto que existem 2 módulos diferentes. Serão desenvolvidas 2 aplicações *Front-End* separadas: uma para a empresa que desenvolveu o software e outra para os utilizadores desse software. Para a comunicação entre estes elementos será usado *chat* em tempo real.

A abordagem deste projeto passa pela análise de soluções já implementadas, para definir as funcionalidades a desenvolver e as que não justificam o tempo de desenvolvimento. Também são analisadas novas funcionalidades que possam trazer valor para o projeto.

1.5 Estrutura do Documento

No primeiro capítulo deste documento é apresentado o problema a resolver, integrado num contexto atual. São definidos os objetivos a cumprir e a abordagem adotada para os cumprir.

No segundo capítulo é apresentado o estado da arte, onde é analisado o contexto teórico do tema em questão. São analisadas algumas soluções já desenvolvidas, onde são avaliadas as funcionalidades e abordagens destas e são escolhidas as que melhor se adequam e resolvem o problema que está em questão. Por fim, são analisadas algumas tecnologias a serem utilizadas no desenvolvimento da solução, sendo que, caso existam dúvidas entre algumas, estas são comparadas e a escolha é justificada.

A Análise de Valor, onde é analisado o valor que a plataforma traz para o cliente, assim como as estratégias de negócio a adotar para maximizar o sucesso do serviço, são apresentados no terceiro capítulo.

No quarto capítulo é realizada uma análise e *design* à solução a implementar. Na análise são definidos os requisitos principais (sendo que podem ser comparados a outras alternativas e justificada a escolha), os atores principais e alguns requisitos funcionais. Posteriormente, na fase de Design, é idealizado todo o desenvolvimento da aplicação. Nesta fase são utilizados artefactos para planear o desenvolvimento e implantação da solução.

No quinto capítulo é abordado o desenvolvimento do sistema, detalhando os requisitos e funcionalidades principais ou mais complexos.

O sexto capítulo descreve o processo de avaliação, onde a solução e todos os seus componentes são testados, avaliados e o resultado desta avaliação é apresentado e analisado.

O último capítulo (capítulo 7 – conclusão) lista os objetivos atingidos, por atingir e resumo o que será necessário desenvolver no futuro.

2 Estado da Arte

Este capítulo tem como objetivo descrever a situação atual, relativamente a tudo o que engloba o projeto. Para uma análise informada e suportada, são utilizados vários estudos e artigos publicados como referência. Para isto, é realizado um processo de recolha e seleção de informação descrito inicialmente. Posteriormente, no Contexto Teórico, é aprofundado o que é um serviço de *help desk* e quais as particularidades de um departamento que preste este serviço. É também feita a análise, avaliação e comparação entre 2 serviços já existentes, onde podem ser retirados conceitos úteis para o projeto a desenvolver.

Finalmente, para as tecnologias mais avançadas, são analisadas e comparadas para, posteriormente, ser tomada uma decisão informada sobre a melhor tecnologia a adotar para o projeto.

2.1 Recolha e Seleção de Informação

Neste capítulo é analisado o estado da arte existente, para estabelecer um contexto mais aprofundado e apoiar a tomada de decisão para vários pontos do projeto. Como tal, é importante garantir que a informação que está a ser analisada é a mais correta possível.

É possível dividir as referências a utilizar em 2 categorias diferentes: Blogs e publicações em Websites, e publicações tradicionais como artigos e livros.

A Internet não é a fonte mais credível de informação, no entanto, é bastante útil devido à sua extensão e quantidade de dados. O processo de levantamento de informação da internet utilizado foi o seguinte:

1. Definição de Palavras chave;
2. Pesquisa através da utilização das palavras-chave definidas;
3. Seleção - analisando os títulos e descrições das publicações;
4. Seleção de citações e informação através da leitura das publicações selecionadas anteriormente.

Avançando para a segunda categoria, nos artigos e livros, a informação é mais credível, pois é escrita num contexto académico ou profissional e é, posteriormente, revista e avaliada. No entanto, também tem as suas desvantagens, como por exemplo, a dificuldade de encontrar informação relevante. Foi utilizado um processo de pesquisa e seleção das melhores fontes, descrito abaixo (é utilizada a pesquisa para a secção 2.2 como demonstração):

1. Definição de Bibliotecas: (ACM, s.d.) (Google Scholar, s.d.);
2. Definição de Palavras-Chave: **Help Desk, Software, Support**;
3. Pesquisa por palavras selecionadas nas bibliotecas definidas;
4. Seleção inicial através de leitura dos títulos e resumos dos artigos**:** (Sperl, 2006)**,** (Evans & Teresa Jones, 2005) (Cruess, 2002) (Pinard, Evans, & Mankovskii, 2001);
5. Seleção final de informação e citações através da leitura dos artigos filtrados anteriormente: (Cruess, 2002)**.**

O processo poderia englobar mais fases, como por exemplo a avaliação das fontes selecionadas. Porém, estas fases não foram utilizadas por não serem consideradas necessárias.

2.2 Contexto Teórico

Para definir o contexto teórico desta aplicação é importante contextualizar a definição de um help desk e o que é pretendido deste serviço.

"It is common practice for Software Suppliers to Set up help deskS So that users of the Software can Seek help to diagnose and Solve computer Software and hardware problems. In most instances, these help deskS are accessible only by telephone over a public Switched telephone network (PSTN)." (Pinard, Evans, & Mankovskii, 2001).

Um serviço de *help desk* fornece um ponto de contacto único, para os seus utilizadores obterem assistência (Techopedia, s.d.).

De um departamento de *help desk* é esperada a capacidade de apoio aos utilizadores ou clientes de um dos seus produtos ou serviços, na utilização do mesmo, estando disponíveis para responder a quaisquer dúvidas que possam surgir e eventuais obstáculos.

Um serviço ou plataforma de *help desk* visa aumentar a eficiência e eficácia deste processo, disponibilizando um ponto de comunicação e registo das perguntas efetuadas pelos utilizadores, assim como uma gestão destas questões internas e a sua distribuição pelos recursos disponíveis.

De acordo com um estudo feito por Alison Cruess, existem 2 formas de categorizar os problemas reportados pelos utilizadores - Gravidade e Prioridade: *"Rate each problem on its severity and priority. Severity is the level of criticality based on the nature of the failure. It remains the same throughout the life of the problem. Use Priority to distinguish between problems of the same severity."* (Cruess, 2002).

As funcionalidades principais de um serviço de help desk são, de acordo com (Patterson, 2017):

- Acessibilidade e facilidade de uso (Plataforma deve ser intuitiva e a sua utilização deve ser simples e concisa);
- Múltiplos canais para envio de tickets (Deve ser permitido criar tickets a partir de múltiplos canais como: e-mail, chat ou chamada telefónica);
- Permitir aos utilizadores rastrear os seus tickets (Utilizadores devem ser capazes de observar o desenvolvimento do seu pedido);
- Proporciona possibilidade de utilizadores pesquisarem respostas às suas questões (páginas de FAQ são muito comuns neste exemplo, em que o próprio utilizador deve ser capaz de encontrar a resposta á sua questão sem interação com um colaborador);
- Sistema de notificações eficiente (Devem existir notificações para as alterações realizadas ao ticket, estas notificações devem ser enviadas apenas quando a ação o justifica);
- Suporta agrupamento de tickets relacionados (Tickets podem ser agrupados por diversas categorias, pelo que é mais simples encontrar tickets que possam ser semelhantes);
- Configurável e Adaptável (sistema deve ser adaptável a qualquer negócio e plataforma, também deve existir configurações para facilitar a adaptabilidade).

2.2.1 B2B e B2C

É importante também definir as principais diferenças entre B2B e B2C, sendo que um sistema de *help desk* pode visar responder aos requisitos de ambos sistemas de negócio.

Uma empresa B2B atua num mercado de nicho, cujo cliente é outra empresa. O tipo de compra é mais racional, envolvendo tipicamente diversos intervenientes e um maior valor. Neste sentido, temos um ciclo de venda mais longo, com compras de maior envolvimento. (Morais, 2018). Sendo assim, um departamento B2B tem requisitos mais complexos e o *help desk* é bastante importante, pelos seguintes pontos:

- Visto ser um produto desenvolvido para uma empresa, tem que responder aos requisitos de negócio dessa empresa o que, por vezes, pode não ser o caso específico e serem necessárias mais alterações ao produto;
- A fase de adoção do produto é bastante importante e pode ser complexa. É crucial que, nesta fase, todos os eventuais problemas sejam devidamente abordados;
- O compromisso é maior, e com isso vem a necessidade de apoio constante;

Por sua vez, uma empresa B2C trabalha para um mercado maior, em que o Cliente é um consumidor individual. Aqui, o tipo de compra é mais emotivo e impulsivo, com poucos ou nenhum interveniente. Por ser uma compra tipicamente de baixo envolvimento, o ciclo de vida é mais curto, mas o valor médio também é menor (Morais, 2018). Este caso, apesar de tipicamente mais simples, também traz os seus problemas, como:

- A plataforma pode não estar preparada para todos os tipos de utilizadores e muitos destes podem ter dificuldade e necessitar de apoio na sua utilização;

- A fase inicial para cada cliente pode ter processos que envolvam suporte por parte da organização (por exemplo: Ao criar conta numa plataforma que envolve transferências bancárias, muitas vezes é necessário um colaborador intervir para validar os documentos e garantir a autenticidade do utilizador).

Ambos os negócios possuem alguns requisitos distintos - uns com mais ênfase do que outros - uma plataforma de *help desk* que não vise focar-se em apenas um deles; tem de ser capaz de corresponder a ambos os tipos de negócios.

2.3 Soluções e Abordagens existentes

Atualmente, já existem alguns serviços de *help desk* idênticos ao que é pretendido com este projeto, muitos destes bem-sucedidos. Devido a isto, é importante avaliar estas soluções para perceber o que pode trazer valor a este projeto e o que pode ser melhorado ou evitado.

Através de duas referências diferentes ((Capterra, s.d.) e (Jonas DeMuro, 2019)) foi percebido que, entre as diversas plataformas existentes, destacam-se duas - quer em número de *reviews*, funcionalidades e rating geral: Freshdesk e Zoho Desk.

2.3.1 Freshdesk

O Freshdesk é uma plataforma de suporte aos clientes (help desk), desenvolvida pela Freshworks. Tratando-se de um *help desk* estabelecido, já vem a resolver algumas das questões que este projeto pretende resolver, como tal, é importante analisar a plataforma para perceber como irá ser possível resolver algumas destas questões e definir a melhor abordagem para tal.

A Freshdesk é uma plataforma focada nas empresas. Nesta plataforma é possível dar suporte aos clientes em diversos projetos e através de vários canais de comunicação, mantendo uma gestão eficiente dos tickets e um processo que visa melhorar as relações das empresas com os seus clientes.

A Freshdesk divide as suas funcionalidades nas seguintes categorias:

- Gestão de Tickets;
- Colaboração;
- Suporte de vários canais de comunicação;
- Automatismos;
- Self-Service para os clientes;
- Dados estatísticos;
- Customização;
- Segurança.

Em relação a estes grupos, serão analisados os mais relevantes para o projeto em questão e levantadas as funcionalidades que podem ser aproveitadas para melhorar e complementar o projeto.

2.3.1.1 Gestão de Tickets

Nesta categoria, a Freshdesk apresenta todos os tickets atuais numa lista única, visível por toda a equipa. É permitido filtrar os tickets por atribuição, prioridade e estado. Na Figura 1 é possível observar a listagem de tickets da Freshdesk:

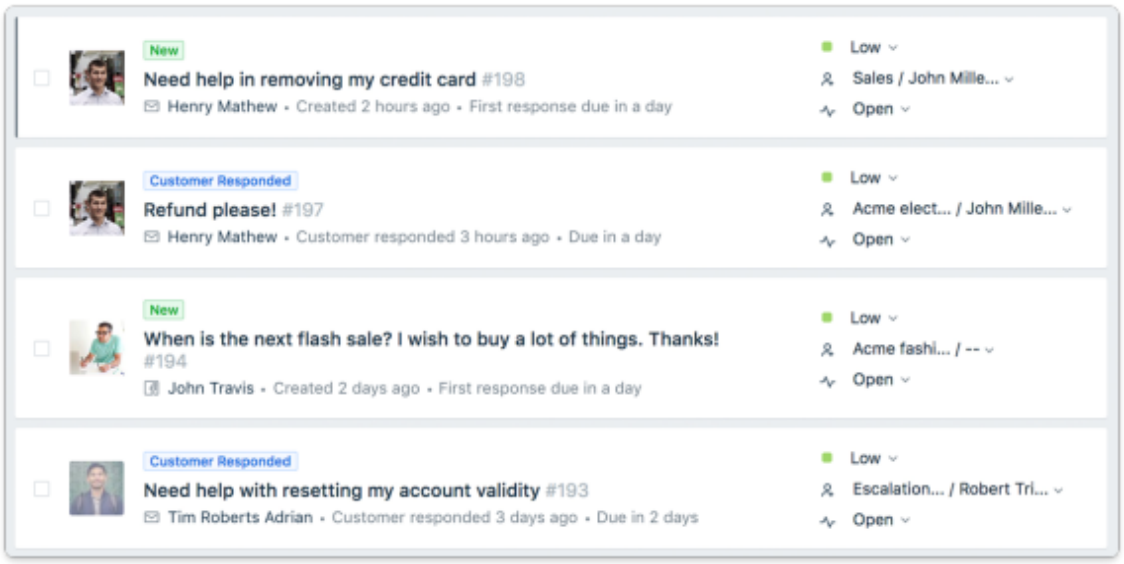


Figura 1 - Listagem de Tickets Freshdesk (Freshworks, Freshdesk, s.d.)

Nesta listagem, é possível observar rapidamente o assunto do ticket, o seu código, a ação (novo ticket/ resposta do utilizador) assim como a prioridade, o departamento e responsável pelo ticket e o seu estado.

Nesta categoria também existem as seguintes funcionalidades:

- Gestão de Prazos para tickets;
- Deteção de colisão entre Agentes;
- Configuração de Estados;
- Possibilidade de efetuar várias ações sobre o ticket rapidamente;
- Respostas pré-formatadas;

2.3.1.2 Suporte de vários canais de comunicação

A Freshdesk permite a criação automática de tickets, através de mensagens enviadas por diversos canais de comunicação como:

- E-mail: É possível integrar um e-mail de suporte: quando este receber algum e-mail um ticket é gerado automaticamente;
- Telefone: É possível gravar conversas telefónicas e convertê-las em tickets;
- Chat;
- Redes Sociais: Tal como no e-mail, é possível integrar uma página do Facebook ou Twitter para gerar automaticamente um ticket, quando é recebida uma mensagem;

- Web Site: A Freshdesk cria um subdomínio para cada empresa que adere ao software, aqui é possível para qualquer cliente criar tickets manualmente.

Em relação ao chat, dentro da plataforma Freshdesk existe uma funcionalidade de chat onde é permitida a conversa com o cliente e posteriormente gerar ou associar a conversa a um ticket.

2.3.1.3 Automatismos

Dentro desta categoria a Freshdesk disponibiliza alguns automatismos e permite a configuração de outros, o que possibilita o aumento da eficiência.

Dentro destes podem ser destacados os seguintes.

- Atribuir tickets a responsáveis dependendo de palavras chave, competências e quantidade de trabalho
- Definir prioridade de tickets automaticamente
- Lembretes de tickets pendentes á algum tempo
- Lançamento de alterações baseadas em ações no ticket (exemplo: Caso um ticket esteja fechado e uma nova mensagem for enviada pelo responsável, reabrir o ticket)
- Envio de e-mails acerca de alterações no ticket

Com estes automatismos a Freshdesk pretende que todo o trabalho feito seja através da lista de tickets atribuídos a cada pessoa e evitar que o processo de escalonamento seja manual ou sejam realizadas tarefas repetitivas.

2.3.2 Zoho Desk

O Zoho Desk é outra plataforma de Help Desk bem estabelecida e bastante utilizada, desenvolvida pela Zoho. As funcionalidades que a Zoho Desk oferece são idênticas às da Freshdesk com algumas diferenças:

- Gestão de Tickets;
- Zia (Inteligência Artificial);
- Self-Service;
- Produtividade de Agentes;
- Automatismos;
- Extensibilidade;
- Análise;
- Segurança.

2.3.2.1 Gestão de tickets

Nesta categoria, a Zoho insere as funcionalidades de comunicação com os clientes através de múltiplos canais e a possibilidade de criar departamentos e produtos dentro da plataforma, para ter os de acordo com a estrutura da empresa.

Através da funcionalidade de chat em tempo real, a Zoho Desk oferece a possibilidade de integrar widgets já desenvolvidos em qualquer plataforma web. Ou seja, com pequenas

alterações a uma plataforma, as empresas que utilizam a Zoho Desk conseguem integrar um chat que os clientes podem utilizar para comunicar com os agentes.

Do lado da plataforma, os agentes têm também um chat para comunicarem com os clientes e aqui poderão gerar ou associar o chat a um ticket.

2.3.2.2 Zia

A Zia é um *bot* desenvolvido pela Zoho, que comunica diretamente com os clientes. O objetivo deste *bot* é evitar a ação humana desnecessária, ou seja, caso um cliente faça uma questão que já foi feita e respondida bastantes vezes a plataforma deve ser capaz de responder sem a ação de um agente. Isto aumenta bastante a eficiência da plataforma pois os agentes podem-se focar nas questões que realmente precisam da sua atenção.

A Zia é também capaz de:

- Utilizar comandos por voz;
- Partilhar soluções relevantes, a partir de uma base de conhecimento;
- Identificar e adicionar ao ticket (como tag) aspetos importantes deste;
- Análise de Sentimentos- Capacidade de analisar se uma mensagem tem um sentimento positivo (agradecimento) ou negativo (queixa);
- Notificações;
- Visualização de Estatísticas;
- Proposta de resposta aos agentes;
- Configuração.

2.3.2.3 Produtividade de Agentes

Aqui, a Zoho oferece várias funcionalidades que aumentam a eficiência dos agentes, tornando o processo de resolver um ticket o mais rápido e eficaz possível.

Neste sentido, existem funcionalidades para a alocação e organização de tickets. Com estas funcionalidades cada agente vai ser atribuído aos tickets, envolvendo o seu espetro de competências e estes vão ser organizados de modo a que sejam destacados os mais atrasados e prioritários. Também é possível, dentro de um ticket, pedir ajuda pela forma de notas a outros agentes, o que aumenta a capacidade de colaboração e entreajuda na empresa e, por sua vez, a eficiência na resolução de um ticket.

Além destas funcionalidades, a Zoho Desk também tem uma aplicação mobile para os agentes poderem responder a tickets, através do telemóvel.

2.3.2.4 Automatismos

Os automatismos definem as ações que o sistema efetua, sem necessidade de ação humana - como notificações automáticas e algumas regras. Para aumentar a eficiência de qualquer plataforma é importante que as ações repetitivas sejam efetuadas pelo próprio sistema e não pelos seus utilizadores.

Para isto, a Zoho Desk oferece várias configurações para ações serem efetuadas diretamente pelo próprio sistema, como por exemplo:

- O que fazer quando um ticket passa o seu prazo?
- O que fazer quando um ticket é terminado?
- Quando um ticket é criado, que prioridade e prazo são atribuídos? Dependendo de que fatores?

Além destes automatismos, também é possível configurar notificações ao cliente e regras de fluxo. As regras de fluxo são ações que podem ser desencadeadas com outras ações humanas, como atualização de campos ou de estados do ticket.

2.3.2.5 Extensibilidade

Nesta categoria, a Zoho desk oferece a possibilidade de integração dos seus serviços com outras plataformas, oferecendo extensões como *widgets* e Serviços Públicos.

Desta forma, caso uma empresa queira desenvolver uma plataforma individual, mas utilizar os serviços da Zoho Desk, é possível.

2.3.2.6 Análise

De uma perspetiva de gestão é importante ter uma visão geral sobre o processo de suporte que a empresa oferece. Para facilitar este processo, a Zoho Desk oferece funcionalidades como por exemplo: Relatórios estatísticos, *Dashboards* e visualização da gestão de tempo gasto em tickets e tarefas.

2.3.3 Resumo e Funcionalidades relevantes

Ambas as plataformas analisadas são bastante complexas e estão bem estabelecidas no mercado, sendo exemplos do que uma plataforma de *help desk* precisa para ser bem-sucedida. Como tal, existem algumas funcionalidades principais que vale a pena destacar, pois podem ser reaproveitadas pelo projeto:

Tabela 1 - Funcionalidades relevantes Freshdesk e Zoho Desk

Funcionalidade	Plataforma	Aproveitamento
Escalonamento de Tickets	Ambas	Caso ticket não seja atribuído a um agente específico faz sentido que a plataforma seja capaz de fazer esse escalonamento automaticamente.
Gestão de Tickets	Freshdesk	Uma gestão eficaz de tickets ajuda a tornar o trabalho dos agentes o mais eficiente possível, uma boa funcionalidade neste sentido seria a possibilidade de existir uma listagem de tickets com filtros e ordenação onde se destacavam os mais prioritários e atrasados.
Chat	Ambas	Ambas as plataformas têm uma funcionalidade de chat onde posteriormente se poderá associar ou gerar um ticket, isto é bastante útil pois permite uma comunicação mais simples e eficaz com os utilizadores.
Automatismos	Ambas	Ações repetitivas como notificações, alterações de estados dos tickets consoante as mensagens e outras a analisar devem ser automatizadas para tornar mais eficiente o trabalho dos agentes.
Extensibilidade	Zoho Desk	Deve ser possível a integração do serviço com outras plataformas externas, ou seja, caso uma empresa deseje desenvolver algo deve ser possível e simples a integração com os serviços desenvolvidos.
Widget	Zoho Desk	A possibilidade de disponibilizar um widget para integração nas plataformas que tratava do chat com os colaboradores de um help desk.

Estas são as funcionalidades mais relevantes para o projeto a desenvolver, como tal alguns dos requisitos podem ser baseados nestas funcionalidades, assim como em alguns aspetos visuais.

2.4 Tecnologias e Arquiteturas Existentes

Será necessária a escolha entre as tecnologias e arquiteturas de desenvolvimento a utilizar. Para uma decisão informada, será feita uma análise às mais relevantes. Neste caso, destaca-se a arquitetura a adotar e, caso esta arquitetura requeira uma comunicação por mensagens, o melhor broker a usar.

2.4.1 Arquitetura

Com o crescimento tecnológico, aumenta também a complexidade e requisitos de performance dos sistemas. Com este aumento surgiu a necessidade de uma melhoria e avanço das

arquiteturas dos sistemas, para que estes possam corresponder aos seus requisitos. Com isto, existem atualmente 3 arquiteturas principais: Monolítica, SOA e de Micro Serviços.

2.4.1.1 Arquitetura Monolítica

"In the traditional application, all the business components are packaged together, distributed and deployed as a whole, this development and deploy pattern is called monolithic" (Zhongshan Ren, 2018).

Monolítico significa que é composto por apenas um serviço ou componente, ou seja, a parte de *back end* do sistema está comprimida dentro do mesmo serviço ao invés de estar dividida em múltiplos

A Figura 2- Arquitetura Monolítica representa a estrutura de uma aplicação monolítica.

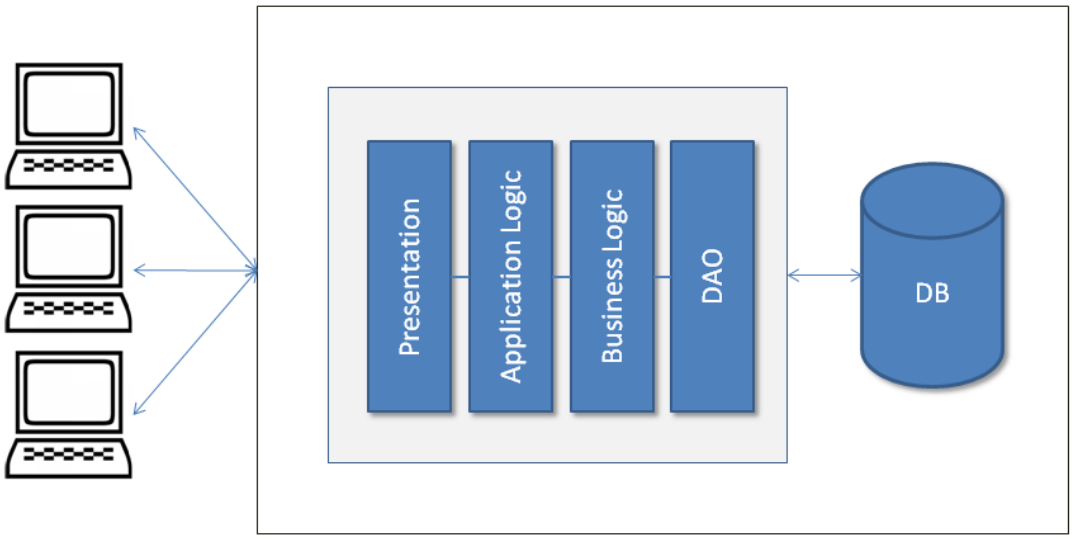


Figura 2- Arquitetura Monolítica

Como se pode observar na Figura 2, retirada de (Limitation of monolithic architecture versus its solution with microservices, 2020), toda a lógica de negócio está comprimida no mesmo serviço. Existem várias vantagens e desvantagens ao adotar uma arquitetura monolítica e é necessária a análise destas para decidir se será ou não uma boa opção para o projeto em causa.

Na Tabela 2 estão descritas as vantagens e desvantagens desta arquitetura:

Tabela 2 - Vantagens e Desvantagens de uma arquitetura Monolítica

Vantagens	Desvantagens
Simplicidade de Desenvolvimento	Manutenção
Simplicidade de Implantação	Dimensão do serviço poderá afetar tempo de *start-up*
Simplicidade para Testar	Necessidade de implantar uma nova versão a cada atualização
Simplicidade para Escalar Horizontalmente	Dificuldade de escalar verticalmente
	Fiabilidade
	Dificuldade de adotar ou integrar novas frameworks ou tecnologias

Através das vantagens e desvantagens, pode concluir-se que esta arquitetura se adequa melhor a aplicações simples e de pequena dimensão, pois todo o processo de desenvolvimento, testes e implantação é mais eficiente. No entanto, à medida que a dimensão da aplicação aumenta ou surge a necessidade de a escalar com novas funcionalidades ou alterações, torna-se desvantajoso o uso de uma arquitetura monolítica.

2.4.1.2 Arquitetura SOA

"*Service-oriented architectures (SOAs) are well established as an architectural paradigm for distributed systems. With software systems becoming more and more complex over time, quality assurance becomes increasingly important*." (Andreas Goeb, 2011).

SOA, ou arquitetura orientada a serviços, é uma arquitetura que permite a vários serviços a comunicação entre eles. Desta forma, permite-se a utilização de diversas linguagens ou frameworks de desenvolvimento, dentro do mesmo sistema, enquanto se mantém um baixo acoplamento entre os diferentes serviços.

Com o SOA são desenvolvidos diversos serviços dependendo dos requisitos do sistema. Geralmente estes serviços são desenvolvidos de acordo com funcionalidades ou módulos principais, sendo que é desenvolvido um serviço por funcionalidade/módulo.

Geralmente, o SOA utiliza um componente para orquestração entre os diversos serviços. Orquestração significa que esse componente está responsável por coordenar e realizar o fluxo de negócio do sistema. Para esta função pode ser utilizado um ESB ou até mesmo um serviço.

A utilização desta arquitetura requer o uso de alguns padrões para assegurar a sua fiabilidade e integridade. Algumas delas são:

- Documentação e definição do padrão dos contratos dos serviços;
- Baixo acoplamento;
- Abstração;
- Reutilização;
- Autonomia.

Na Tabela 3 estão as vantagens e desvantagens desta arquitetura.

Tabela 3- Vantagens e Desvantagens de SOA

Vantagens	Desvantagens
Reutilização de Serviços	Sobrecarga Alta
Fácil manutenção	Alto investimento inicial
Independente de plataformas (linguagens de programação/*frameworks*)	Gestão complexa dos serviços
Disponibilidade	
Fiabilidade	
Escalabilidade	

Pode concluir-se, que SOA é mais adequado para aplicações com uma maior dimensão e com fluxos bem definidos, sendo que também se torna desvantajoso a gestão deste fluxo e dos serviços em si.

2.4.1.3 Arquitetura de micro serviços

"*The microservice architecture is a lightweight, miniaturized development and operation mode. Each microservice deployed independently and follow the single responsibility principle, different microservice can use different technology stack, and update independently. Microservices use lightweight HTTP protocol for communication and data exchange. For applications, which need to realize the high-concurrency and high-capacity system, microservices technology architecture is a good choice*." (Zhongshan Ren, 2018).

Micro Serviços são Serviços pequenos, desenvolvidos para responder a uma necessidade bastante coesa e específica. Por isso, uma arquitetura de Micro Serviços é uma arquitetura baseada no desenvolvimento de vários destes serviços, cada um deles sendo responsável por uma única parte do negócio e independente dos demais. Desta forma, especificidades como a linguagem adotada ou frameworks,não influenciam os outros serviços nem a plataforma no seu todo.

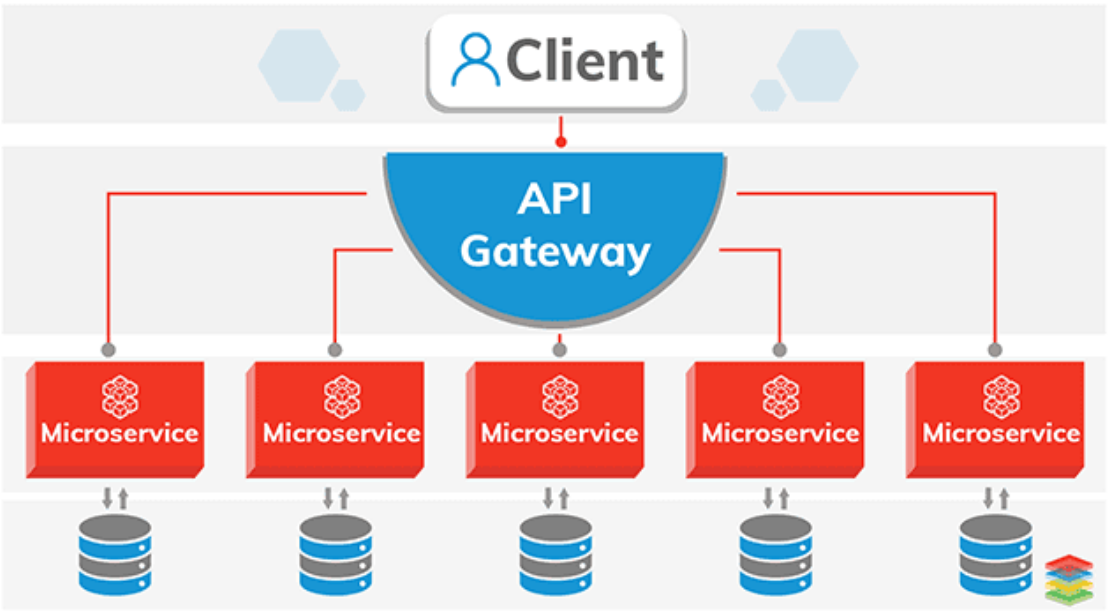


Figura 3 - Exemplo de Arquitetura de Micro Serviços

Na Figura 3, retirada de (Microservices Architecture Design and Best Practices, 2018), pode-se observar um diagrama que representa esta arquitetura. Percebe-se que a lógica do negócio está dividida entre vários micro serviços e que cada um destes tem uma base de dados independente, o que também é uma das boas práticas desta arquitetura (*database per service*).

Outra caraterística que salta á vista na Figura 3 é o *API Gateway*. O Cliente não precisa de saber como se tem divididas as responsabilidades do lado do *Back End* e, como tal, estes serviços devem estar desacoplados entre eles mesmos e entre o próprio cliente. Uma das soluções para este problema é a utilização de um Gateway. Um Gateway funciona como intermédio entre o cliente e a estrutura de serviços da plataforma, como tal, utiliza HTTP ou HTTPS para redirecionar um pedido do cliente para o/os serviço/serviços corretos. Através da utilização de um *gateway*, qualquer alteração a um serviço não afetará o cliente e estes poderão ser escalados e alterados sem qualquer impacto no funcionamento do sistema. Além destes pontos o *Gateway* também pode ser utilizado para autorização e registo de pedidos (*logging*).

No entanto, além da comunicação com o Gateway, os serviços também podem necessitar de comunicar entre si. Neste caso, será HTTP a melhor opção? Na grande parte das vezes não! O objetivo desta arquitetura é que os serviços sejam completamente desacoplados uns dos outros, para isso, quando um serviço é alterado, não tem de levar em conta a comunicação entre outros serviços, nem pode precisar desta para a realização de qualquer função, logo, HTTP não seria o ideal. Aqui entra a comunicação por eventos (*Event-Driven*) através de *messaging*.

Event-Driven é uma arquitetura que funciona por publicação de eventos num canal para comunicação ente vários serviços. Esta arquitetura funciona da seguinte forma: Um serviço publica um evento num canal específico, neste canal há outros serviços “à escuta” de eventos. Desta forma, existe um produtor e um recetor de eventos. Após um evento ser detetado pelo

recetor, é analisado e tratado assincronamente e são realizadas (caso seja necessário) as ações necessárias.

Existem diversas tecnologias para desenvolver comunicação através de *messaging*, sendo que as duas mais utilizadas são Apache Kafka e RabbitMQ.

Na Tabela 4 pode-se observar as principais vantagens e desvantagens desta arquitetura:

Tabela 4- Vantagens e Desvantagens da Arquitetura de Micro Serviços

Vantagens	Desvantagens
Agilidade	Complexidade Arquitetural
Equipas pequenas e especializadas	Desenvolvimento e testes (Integração)
Pouco código (complexidade baixa em cada serviço)	Performance
Possibilidade de uso de diversas tecnologias	Integridade dos dados
Escalabilidade	Gestão
Isolação de Dados	Versionamento
	Competências técnicas exigentes

2.4.1.4 Comparação

Como foi analisado, existem várias vantagens e desvantagens na utilização de cada uma das arquiteturas. Na Tabela 5 é analisada cada possível vantagem ou desvantagem e como a arquitetura em questão responde a esta.

Tabela 5- Comparação de Arquiteturas

Ponto	Monolítica	SOA	Micro Serviços
Escalabilidade	Difícil de escalar, para cada nova funcionalidade é necessária a alteração do serviço completo e a implantação do mesmo.	Relativamente fácil de escalar, é apenas necessário alterar o serviço que trata dessa funcionalidade (ou criar um novo). E atualizar o Orquestrador.	Fácil de escalar, sendo apenas necessária a alteração do micro serviço em causa (ou criar um novo)
Suporte para diversas tecnologias	Não suporta	Suporta	Suporta
Desacoplamento entre componentes	Não suporta	Os serviços desenvolvidos são independentes e desacoplados entre eles, no entanto têm um componente que os coordena.	Os serviços desenvolvidos são independentes e desacoplados entre eles funcionando individualmente.

Divisão de Tarefas entre equipas	Cada equipa trabalhar no mesmo serviço, por isso tem de conhecer o negócio como um todo e ser especializada neste.	Cada equipa pode ser especializada no seu serviço, sendo que os serviços podem ser relativamente grandes e por isso divididos entre várias equipas, também é necessário o conhecimento do fluxo de negócio.	Cada equipa é especializada no seu próprio serviço e pode funcionar independentemente das restantes equipas.
Complexidade no Desenvolvimento	Complexa, um serviço trata do negócio todo e por isso tem de tratar de cada funcionalidade.	Relativamente simples, cada serviço trata de uma parte do sistema e como tal deve ser simplificado.	Simples, cada serviço trata de uma pequena parte independente do sistema, como tal deve ser bastante simples.
Complexidade de Arquitetura	Simples, existe apenas um serviço.	Complexa, existe um serviço para cada grande módulo do sistema e estes são orquestrados por outro componente.	Complexa, existem diversos serviços, sendo que cada um tem de ser desacoplado dos outros, mas mesmo assim manter todas as necessidades e fluxos do negócio.
Testabilidade	Simples, existindo apenas um serviço os testes são bastante localizados e existem poucos testes de integração.	Relativamente complexa, são necessários testes de integração entre o orquestrador e os serviços.	Complexa, são necessários testes de integração entre todos os serviços, todos os eventos e a sua receção também têm de ser testados.
Fiabilidade	Fiável.	É necessário um bom design e análise para ser fiável, caso contrário pode haver problemas de integração.	É necessário um bom design e análise para ser fiável, caso contrário pode haver problemas de integração.
Investimento inicial	Baixo, Serviço vai sendo aumentado conforme as necessidades.	Alto, toda a arquitetura e os vários serviços têm de ser desenvolvidos desde o início	Alto, toda a arquitetura e os vários serviços têm de ser desenvolvidos desde o início

Como se pode observar, a arquitetura monolítica separa-se bastante das outras duas, sendo que estas são bastante semelhantes. De certa forma, pode considerar-se micro serviços como uma melhoria de SOA, sendo que os serviços são ainda mais pequenos e desacoplados.

2.4.2 Message Broker

RabbitMQ e Apache Kafka são duas das frameworks mais populares e modernas para *Messaging e Event-Driven Arquitecture,* no entanto, não são idênticos. Ambos têm fluxos diferentes e podem ser utilizados de maneiras diferentes pelo que, as necessidades e funções de cada sistema, determinam a melhor framework a adotar.

2.4.2.1 RabbitMQ

Bastante utilizado e bem suportado (com bibliotecas de integração em Java, .NET, PHP e muitas outras), RabbitMQ é um Message Broker bastante tradicional e simples. Como se pode observar na Figura 4, retirada de (Humphrey, 2017) o RabbitMQ tem os seguintes componentes principais:

- *Producer*-> Serviço que cria e envia a mensagem para um *Exchanger*;
- *Exchanger*-> Serviço responsável para alocar a mensagem á *Queue* correta;
- *Queue*-> Componente responsável por guardar e disponibilizar as mensagens na ordem recebida;
- *Consumer* -> Serviço que consome uma mensagem.

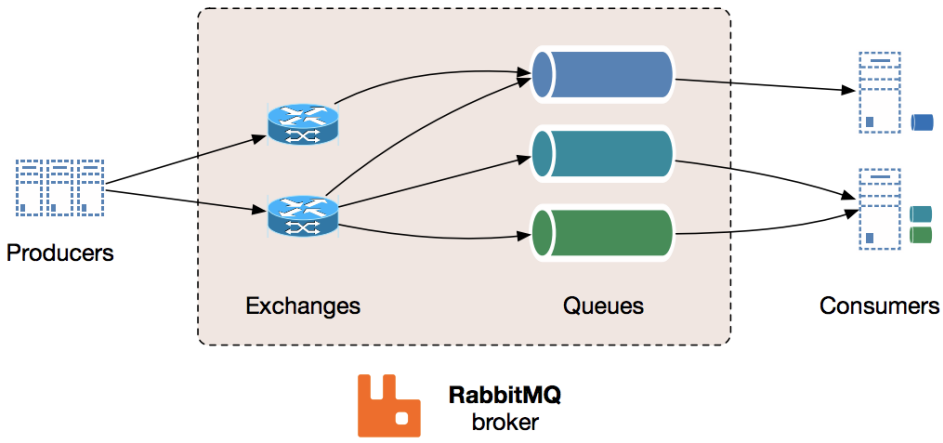


Figura 4 - Diagrama de RabbitMQ

Desta forma, um *producer* envia uma mensagem para um *exchanger* que por sua vez a aloca à *queue* correta. Posteriormente, os *consumers* que estão a ler dessa *queue* obtêm a mensagem. Uma das particularidades de RabbitMQ é a utilização *exchangers* para desacoplar a necessidade de conhecer as filas ou os seus roteamentos do produtor.

Na Tabela 6 são avaliadas as vantagens e desvantagens desta framework.

Tabela 6- Vantagens de Desvantagens de RabbitMQ

Vantagens	Desvantagens
Pouca complexidade	Pode haver problemas caso exista uma grande quantidade de mensagens (Performance)
Simples de manter	Não há replicação de mensagens, se um queue for abaixo as mensagens podem ser perdidas
Bibliotecas disponíveis na maior parte das linguagens principais	Apenas um *consumer* pode processar uma mensagem de uma *queue*, caso existam vários *consumers* as mensagens têm de ser divididas por várias *queues*
Interface de monitorização embutida	Não ordena mensagens

2.4.2.2 Apache Kafka

Sendo construído na *LinkedIn*, o Apache Kafka foi desenvolvido para resolver uma variedade de problemas relativos à disponibilização de mensagens com grande volume, a um grande número de recetores (Guozhang Wang, et al., 2015).

Apache Kafka foi desenhado tendo em conta sistemas mais complexos e com um fluxo alto de mensagens, para garantir a rapidez e escalabilidade do sistema. Provisiona um armazenamento das mensagens durante um longo período.

Ao contrário do RabbitMQ, o Kafka utiliza recetores inteligentes em vez de produtores, ou seja, os recetores é que são responsáveis por rastrearem as mensagens que lhes pertencem e a sua localização.

"With this goal in mind, in 2010 we implemented Kafka as LinkedIn's centralized online data pipelining system. Kafka organizes messages as a partitioned write-ahead commit log on persistent storage and provides a pull-based messaging abstraction to allow both real-time subscribers such as online services and offline subscribers such as Hadoop and data warehouse to read these messages at arbitrary pace. Since Oct. 2012, Kafka has become a top-level Apache open source software and be widely adopted outside LinkedIn as well" (Guozhang Wang, et al., 2015)

Uma mensagem em Kafka possui uma chave, um valor, data e hora.

No diagrama da Figura 5, retirada de (Apache Kafka - Cluster Architecture, s.d.), pode-se observar a estrutura de Apache Kafka e identificando os seguintes componentes:

- *Producer*-> Serviço que cria e envia a mensagem;
- *Broker*-> Componente responsável por guardar as mensagens;
- *Cluster*-> Conjunto de Brokers
- *Zookeeper*-> Componente responsável pela gestão das mensagens nos Brokers e coordenação destas entre os *producers e consumers*.

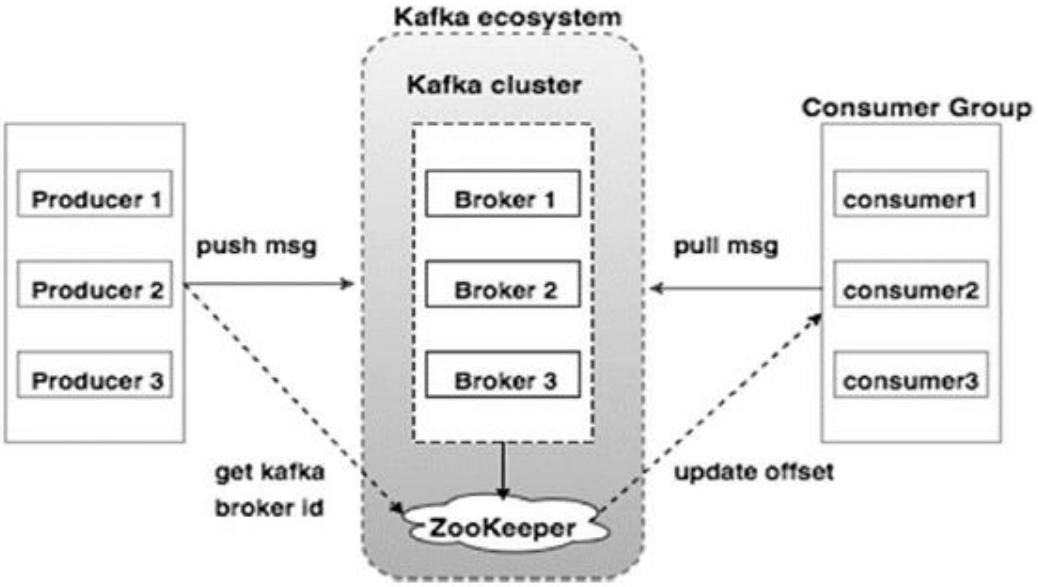


Figura 5 - Diagrama de Apache Kafka

No entanto, há algo que não é retratado no diagrama da Figura 5: os tópicos. Pode-se pensar num tópico como um agrupamento de mensagens e como o assunto dessas mensagens, um exemplo seria o de um veículo, poderia existir um tópico "Veículo" e todas as mensagens sobre os veículos seriam publicadas nestes tópicos. Em Kafka, um tópico pode ser repartido em partições para melhorar a eficiência do sistema. É a um tópico que os *consumers* subscrevem.

Visto isto, o fluxo do Kafka funciona da seguinte forma:

- Um *producer* pretende enviar uma mensagem para um tópico, primeiro obtém (através do *zookeper*) o broker para o qual enviar a mensagem e submete essa mensagem;
- O *broker* guarda essa mensagem dentro do tópico (ou numa das partições);
- O *consumer* subscreve a este tópico;
- O *zookeeper* informa o *consumer* da sua localização nesse tópico (sendo que esta localização aponta para a última mensagem lida);
- O *consumer* faz pedidos aos tópicos repetidamente;
- Após uma mensagem ser publicada, o Kafka reencaminha esta para o *zookeeper,* este depois informa o *zookeeper* que processou a mensagem e o Kafka atualiza a localização deste *consumer*.

Desta forma, caso algo no fluxo falhe, o *consumer* pode tentar processar a mensagem novamente, pois esta ainda estará disponível.

Na Tabela 7 são avaliadas as vantagens e desvantagens desta framework.

Tabela 7- Vantagens e Desvantagens de Apache Kafka

Vantagens	Desvantagens
Performance	Complexidade
Replicação, recuperação e ordenação de mensagens	Não existem bibliotecas para integração com linguagens de programação sendo necessário desenvolver tudo de raiz.
Vários consumers podem consumir a mesma queue/tópico, a mensagem é reencaminhada para todos eles.	Necessita de softwares externos para monitorização

2.4.2.3 Comparação

Ambas as frameworks são ótimas no que fazem, mas têm abordagens diferentes. Enquanto que o RabbitMQ é simples e fácil de implementar, o Kafka preocupa-se mais com a performance e funcionalidade do sistema.

Devido a isto, o RabbitMQ torna-se melhor se estiver a ser implementado um sistema simples de messaging que não tenha muita concentração de mensagens e poucos consumidores. Mas perde a relevância se for necessário um controlo melhor do sistema ou existirem vários *consumers*.

Caso esteja a ser desenvolvido um sistema mais complexo, que espera uma grande quantidade de mensagens e utilização, aí será melhor utilizar o Kafka. Com o Kafka é possível um melhor controlo do que acontece no sistema e a garantia que a performance se mantém a alto nível.

2.5 ChatBots

ChatBots são b*ots* desenhados para conseguir conversar com um utilizador através de um chat. Existem várias organizações que já usam versões deste tipo de *bots* para comunicar com os seus clientes e responder a perguntas que estes possam ter ou ajudá-los a navegar numa plataforma.

O Objetivo principal destes *bots* é diminuir os recursos utilizados pela empresa no suporte a utilizadores enquanto, simultaneamente, tenta oferecer uma experiência imediata e real ao utilizador.

Muitos destes *bots* utilizam inteligência artificial para saber o que responder aos utilizadores e serem o mais prestáveis possíveis.

“As customers spend more time in digital environments, brands are moving into digital services. Technological advances now allow virtual service agents or “e-service agents” to enhance customer experiences and fulfill expectations through real-time interactions”. (Hagberg, Sundstrom, & Egels-Zandén, 2016).

Cada vez mais são utilizadas diferentes versões destes *bots* para substituir ou tornar mais eficiente o departamento de suporte de uma organização. Como tal, é importante analisar

algumas das tecnologias com que é possível desenvolver um *chatbot*, para perceber qual seria a melhor a adotar. De um estudo realizado por (Rehan, 2019), foram retiradas as duas melhores frameworks para desenvolvimento de um *chatbot,* sendo estas:

- Microsoft Bot Framework;
- Wit.ai.

2.5.1 Microsoft Bot Framework

Desenvolvida pela Microsoft, esta framework de desenvolvimento de *chatbots* é uma das mais utilizadas, por ser possível e simples a sua integração com canais de comunicação já existentes como (Slack, Messenger, Office 365, entre outros). De acordo com (ActiveWizards), esta framework está dividida em dois componentes:

- A própria framework: O SDK (*Software Development Kit*) responsável pelas integrações básicas do *bot;*
- LUIS.ai (Language Understanding Inteligent Service): Possuindo reconhecimento de entidades e um sistema de treino, este componente é responsável pela inteligência artificial componente "humana" do *bot*.

Uma das maiores vantagens desta framework é a sua possível integração com diversas plataformas. Mesmo assim, a Microsoft também disponibilizou a possibilidade de partilha de *bots*, através do *website* (https://bots.botframework.com).

De acordo com o mesmo estudo, estas são as principais vantagens e desvantagens desta framework:

Tabela 8- Vantagens e desvantagens da Microsoft Bot Framework

Vantagens	Desvantagens
Ótima para a construção de *bots* assistentes;	Código demasiado complexo;
Suporte pela comunidade de programadores independentes (plataforma de ".Net");	Demasiado dependente dos serviços da Microsoft, não é possível o *deploy* fora de plataformas da Microsoft;
Fácil integração com plataformas de comunicação;	Fácil de utilização para aplicações simples, mas difícil para plataformas com complexos mais exigentes;

2.5.2 Wit.ai

Primeiramente, desenvolvida como uma plataforma onde qualquer programador fosse capaz de facilmente construir o seu próprio *bot*, esta framework acabou por ser comprada pela Facebook e adotada como fundação para a construção da "Facebook´s Bot API".

Esta framework funciona através do desenvolvimento de "stories", simplificando, é configurado um conjunto de possíveis pedidos e interações com os utilizadores, caso estes façam um pedido parecido ao configurado, o *bot* consegue adaptar-se e responder acertadamente.

De acordo com o estudo de (ActiveWizards), esta framework tem as seguintes vantagens e desvantagens:

Vantagens	Desvantagens
Definição de "stories";	Interpretar mal entidades que são parecidas com outras e interpretar mal o fluxo do diálogo;
Cargos de Entidades;	"Stories" não são totalmente desenvolvidas
Aprendizagem com exemplos;	
Possui uma "caixa" com registos dos pedidos dos utilizadores e como a framework interagiu com os mesmos;	
É capaz de ser integrada com ferramentas de comunicação como Slack, Messenger e Skype;	

2.5.3 Conclusão

Ambas as plataformas possuem ótimas ferramentas que certamente iriam trazer valor para a solução. Talvez a melhor a adotar fosse a Microsoft Bot Framework, devido ao facto de ser desenvolvida e facilmente adaptada pelas tecnologias que vão ser utilizadas para a solução.

Um *chatbot* seria uma funcionalidade bastante importante para o sistema e que certamente traria maior valor, devido a poupar recursos e pelos utilizadores obterem uma resposta imediata às suas perguntas, caso não fosse possível responder, o próprio *bot* poderia criar um ticket pelos utilizadores, que seria escalado para um colaborador.

3 Análise de Valor

Nesta secção é analisado o valor da solução a desenvolver. Primeiramente, é utilizado o modelo de análise *Front End.* Com esta análise é possível perceber a relevância do desenvolvimento desta solução e a inovação que a mesma pode trazer. São também geradas, analisadas e selecionadas ideias para funcionalidades que o sistema pode ter.

Posteriormente, é analisado o valor que a plataforma traz em várias perspetivas. Seguido pelo modelo Canvas, que visa definir os principais componentes de negócio da solução.

3.1 Análise de inovação *Front End*

"The Front End (FE) is considered as the first stage of new product development, which roughly concerns the period from the idea generation to its approval for development, or its termination" (Dewulf, 2013).

Esta análise de *Front End situa-se na fase inicial de um projeto e antes do início do seu desenvolvimento,* é utilizada para avaliar uma ideia e perceber a sua relevância e se vale ou não a pena avançar com a mesma. De acordo com Crawford e Di Benedetto (Crawford, CA, & Benedetto, 2005), o processo de FEI (*Front End of Innovation*) deve responder às perguntas: O Quê? Porquê? Quem? Quando? Como?

Uma boa análise nesta fase também poderá trazer novas ideias e objetivos para o produto a desenvolver e, dessa forma, assegurar que este terá sucesso numa perspetiva de negócio.

Para a realização desta análise é utilizado o modelo de NCD (New Concept Development), apresentado por Peter Koen e representado na Figura 6, retirada de (Koen, et al., 2001).

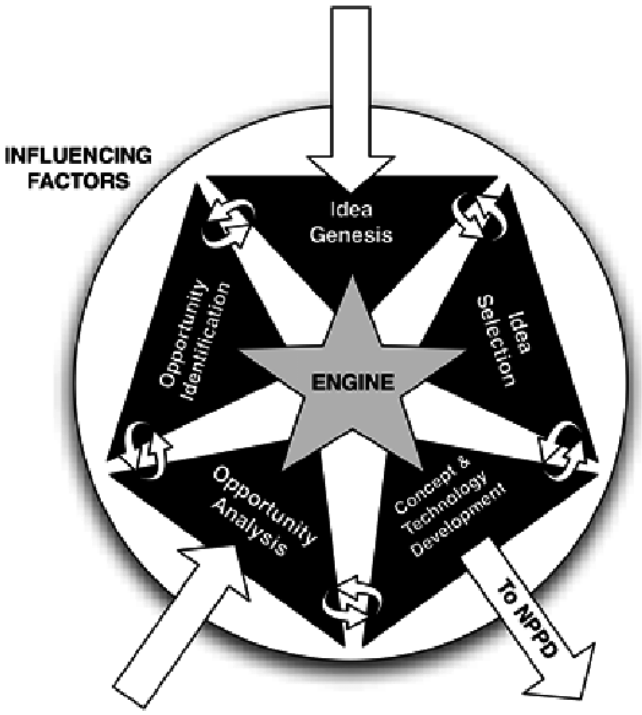


Figura 6- NCD Model

O Motor (*Engine*) representa a liderança e estratégias culturais e de negócio da organização que geram os restantes cinco elementos chave (Dewulf, 2013). Estes 5 elementos chave serão descritos e aplicados ao projeto em causa, nas secções seguintes:

3.1.1 Identificação de Oportunidades

Neste elemento do NCD, são identificadas as oportunidades de negócio existentes.

Existe um grande crescimento na criação de plataformas tecnológicas. Cada vez mais se vê organizações a criarem plataformas e serviços, que visam resolver os diversos problemas e oportunidades do mercado.

Mesmo nas plataformas já desenvolvidas, para se manterem, tem de haver algo que as distinga das demais e muitas organizações percebem que um bom suporte aos utilizadores é uma das formas de o conseguir.

Analisando as plataformas já existentes (cf. Secção ・), percebe-se que estas se focam bastante no processo de help desk interno de cada organização, mas não na interação com o utilizador ou na capacidade de este conseguir observar o estado das suas questões (*tracking*).

Neste sentido, as principais oportunidades encontradas são:

- Crescimento na criação de novas plataformas;
- Necessidade de um bom sistema de suporte nas plataformas já existentes;

- Falta no mercado de um sistema que provisione *tracking* dos tickets aos utilizadores.

3.1.2 Análise de Oportunidades

Neste elemento do NCD pretende-se aglomerar as oportunidades identificadas, no sentido de as traduzir em oportunidades tecnológicas ou de negócio para a organização (Dewulf, 2013).

Nesse sentido, pode-se concluir que cada vez mais irá haver a necessidade de um bom sistema de help desk com funcionalidades que faltem aos outros. Um dos aspetos mais importantes para o funcionamento de uma plataforma é a satisfação dos utilizadores, pelo que se mostra necessário um sistema de help desk focado nos utilizadores e nas necessidades destes.

3.1.3 Geração de Ideias

A geração de ideias é um elemento caraterizado pela elaboração de um brainstorming de onde surgem ideias que podem ou não trazer valor ao projeto em causa. Neste sentido chegou-se às seguintes ideias:

1. Plataforma de tracking de tickets para os utilizadores;
2. Bot para conversa inicial com utilizadores;
3. Automatismo de distribuição de tickets entre departamentos com base em *keywords* e *tags* (baseado em análises prévias de distribuição);
4. Integração com sistemas de Version Control (Team Foundation Services, GitHub, GitLab, BitBucket, etc) para tickets alusivos a um departamento de desenvolvimento;
5. Dashboard com diversas estatísticas apresentadas á base de gráfico para administração
6. Possibilidade de configurar sistema de notificações e e-mails;
7. Em cada ticket, possibilidade de acrescentar notas internas podendo ou não referenciar outros colaboradores para apoio no mesmo;
8. Possibilidade de utilizadores lançarem uma notificação no ticket para alertar o colaborador para o mesmo.

3.1.4 Seleção de Ideias

“Normally there are more opportunities and ideas than can be supported with the funding and time available within the company. The critical activity is to choose which ideas to pursue in order to achieve the most business and consumer value” (Dewulf, 2013).

Sendo assim, neste elemento do NCD são selecionadas as ideias obtidas na secção anterior, sendo filtradas para definir as que serão aplicadas no projeto.

Para tomar esta decisão, irá ser aplicado o método AHP (*Analytic Hierarchy Process*), um método de decisão multi-critério. Este método é utilizado para obter a ideia ideal a desenvolver,

no entanto, como podem ser desenvolvidas várias ideias, pode-se definir que vão ser implementadas as 3 funcionalidades com maior qualificação.

O método AHP tem 7 fases que serão descritas e aplicadas a seguir.

3.1.4.1 Construção da Árvore Hierárquica de decisão

Nesta fase é pretendida a definição do problema e a definição de critérios que irão servir para analisar as diversas soluções, construindo uma árvore hierárquica de decisão que contém o problema (primeira linha), os critérios (segunda linha) e as alternativas (terceira linha).

A árvore hierárquica de decisão para este projeto está representada na Figura 7, nota-se que as ideias contêm a mesma numeração apresentada na secção Geração de Ideias3.1.3.

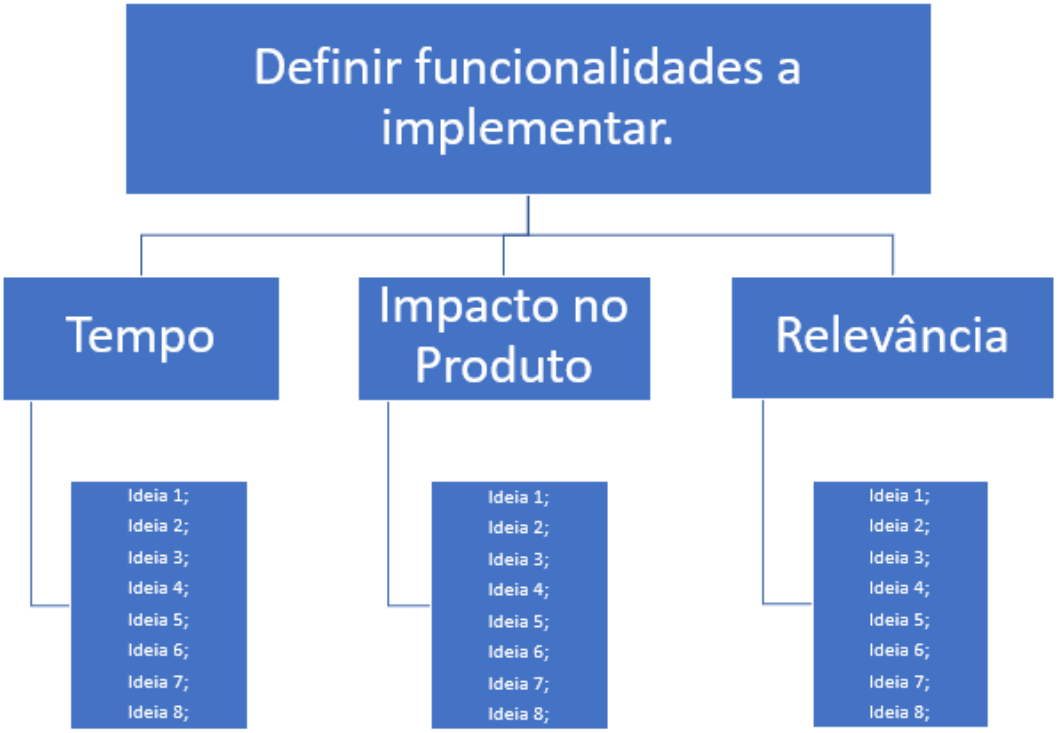


Figura 7 - Árvore Hierárquica de Decisão

3.1.4.2 Comparação de Critérios

Todos os critérios não possuem a mesma relevância para o sistema, para isto é necessário comparar os diversos critérios para definir quais os mais importantes entre eles.

Para isto, é elaborada uma matriz de comparação. Nesta, a importância de cada critério é dada em função de cada um dos outros critérios. A matriz pode ser observada na Tabela 9.

Tabela 9 - Matriz de Comparação de Critérios

Critério	Tempo	Impacto	Relevância
Tempo	1	3	3
Impacto	1/3	1	1
Relevância	1/3	1	1

Com esta matriz pode-se concluir que o critério tempo é 3 vezes mais importante que ambos os outros, que são igualmente importantes entre eles.

3.1.4.3 Definir a prioridade relativa de cada critério

Após a definição da matriz de comparação, é necessário atribuir uma pontuação de importância a cada critério. Para isto, em primeiro lugar, é necessário normalizar a matriz de comparação da Tabela 9, dividindo cada valor pelo total da sua coluna - que pode ser observada na Tabela 10. Após a normalização, a média de cada linha representa a importância de cada critério.

Tabela 10 - Matriz normalizada de comparação de critérios

Critério	Tempo	Impacto	Relevância	Importância
Tempo	3/5	3/5	3/5	**0.6**
Impacto	1/5	1/5	1/5	**0.2**
Relevância	1/5	1/5	1/5	**0.2**

3.1.4.4 Avaliar a consistência das prioridades relativas

Nesta secção pretende-se confirmar que os critérios foram bem definidos e que a importância resultante é relevante, no entanto, devido aos critérios serem relativamente simples e 2 deles serem iguais foi decidido que não era necessária a realização deste passo.

3.1.4.5 Construção da matriz de comparação para cada critério

Nesta secção do AHP é construída uma matriz de comparação das ideias para cada critério, ou seja, para o critério tempo são comparadas as diversas ideias, da mesma forma como foram os critérios anteriormente. No entanto, devido à quantidade de ideias ser bastante grande, este processo foi simplificado, sendo que, ao invés de fazer as 3 matrizes e calcular as prioridades, são definidas as classificações das ideias para cada critério diretamente. Estas classificações são dadas numa escala de 0 a 10.

Neste sentido, as classificações são as apresentadas na Figura 8:

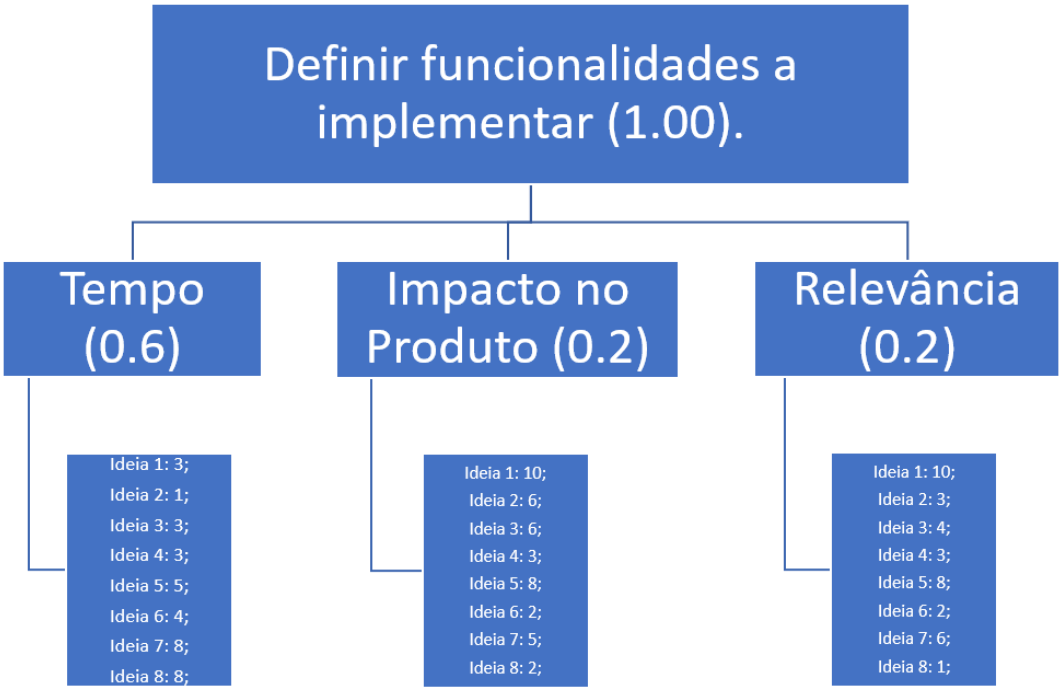


Figura 8 - Classificações de Ideias

3.1.4.6 Obter a prioridade para cada alternativa/ideia

Nesta fase pretende-se obter uma prioridade final para cada ideia, para isto é feito o seguinte cálculo:

$$p1 = (tempo1 * tempo) + (impacto1 * impacto) + (relevância1 * relevância)$$

Ou seja, para a ideia 1, a classificação será: $p = (3 * 0.6) + (10 * 0.2) + (10 * 0.2)$.

Na Tabela 11 estão presentes as classificações de cada ideia:

Tabela 11 - Prioridade das Ideias

Ideia	Classificação
1	5.8
2	2.4
3	3.8
4	3
5	6.2
6	3.2
7	7
8	5.4

3.1.4.7 Escolha da alternativa

Mesmo os valores não sendo percentagens, como seriam se fosse seguido o método AHP no seu todo, continuam a estar ordenados por prioridades. Sendo assim, pode-se definir que irão ser realizadas as 3 funcionalidades com maior prioridade, sendo essas:

- Ideia 1: Plataforma de Tracking de tickets para os utilizadores (5.8);
- Ideia 5: Dashboard com diversas estatísticas apresentadas á base de gráficos para administração (6.2);
- Ideia 7: Em cada ticket, possibilidade de acrescentar notas internas podendo ou não referenciar outros colaboradores para apoio no mesmo (7).

3.1.5 Desenvolvimento de Conceitos e Tecnologias

Neste elemento final é feita a consolidação das ideias e definição concreta do produto a desenvolver.

Assim sendo, o produto será um serviço de *help desk* focado tanto no processo interno como no utilizador e na sua satisfação. Este serviço será dividido em 2 plataformas distintas, uma para os utilizadores colocarem as suas questões e acompanharem a sua resolução, assim como uma plataforma para a gestão interna destas questões, na organização em questão.

3.2 Valor

Nesta secção pretende-se a definição do valor do produto/serviço a desenvolver.

“What is value?

Different customers will answer to that question in different ways. The value of a product can be the performance of its functions or its aesthetic beauty, when applicable and needed. As a general statement high level performances, capabilities, emotional appeal, style, all compared to cost is commonly what we consider as value.” (Tosca, 2018).

Existem várias interpretações sobre a definição de valor, mas resumem-se à definição das vantagens e benefícios que um produto traz para quem o utiliza.

Seguindo este conceito, pode-se definir a proposta de valor deste produto como a possibilidade de melhorar e informatizar um serviço de help desk de qualquer plataforma, tendo uma gestão interna eficaz e uma interação eficiente e apelativa com os utilizadores.

3.2.1 Valor para o Cliente

"Customer value is the satisfaction the customer experiences (or expects to experience) by taking a given action relative to the cost of that action." (Mansfield, 2018).

De acordo com esta afirmação de Mansfield, o valor para o cliente define o que este espera obter da plataforma antes mesmo de tomar qualquer decisão, ou seja, é o que o cliente espera da plataforma.

Existem 2 perspetivas diferentes no valor que esta plataforma traz para os seus clientes: existe a perspetiva de valor para a empresa que usa a plataforma, e, dentro desta, existe também a perspetiva de valor acrescentado para os clientes desta empresa que também vão beneficiar com as funcionalidades proporcionadas.

Com a adoção da plataforma, uma empresa pode esperar o seguinte:

- **Melhoria de processos internos**

Esta plataforma vai permitir às empresas uma gestão eficaz dos tickets apresentados pelos utilizadores, assim como a capacidade de os distribuir pelos recursos existentes com um simples clique. Os colaboradores vão ser capazes de ter um acesso simples aos seus tickets e ser capazes de conversar com os utilizadores e resolver o problema rapidamente. Com estas funcionalidades, o processo de suporte a clientes será bastante mais rápido e serão eliminados os obstáculos principais, como dificuldades de comunicação e má distribuição de tarefas.

- **Melhoria no sistema de suporte a utilizadores**

Com a possibilidade de aceder a uma plataforma para colocar perguntas e pedir apoio ao departamento de suporte, torna-se muito mais simples este procedimento por parte dos utilizadores que conseguem rapidamente colocar as suas questões e observar o estado destas, invés de ter de efetuar chamadas telefónicas repetidas ou e-mails que não se sabe quando vão ser respondidos. Com isto, visa-se aumentar a satisfação dos utilizadores com a plataforma assim como a sua usabilidade.

- **Crescimento do valor das plataformas**

Com a utilização deste serviço de help desk, o valor de qualquer plataforma é aumentado, pois apesar das vantagens a nível interno, o principal foco é o apoio e ajuda aos utilizadores, que são quem compra e usa a plataforma. Neste sentido, oferecer um serviço de help desk contínuo para qualquer problema que o utilizador possa encontrar, é uma mais valia que pode fazer a diferença na escolha do cliente.

3.2.2 Valor percetível

Nesta secção é apresentado o valor que o utilizador obtém tendo em conta a sua perspetiva do que a plataforma oferece e o que este recebe e utiliza em função dos sacrifícios realizados.

Este valor pode ser distribuído numa perspetiva temporal, seguindo a posição temporal do valor. De acordo com (Woodall, 2003) existem 4 pontos temporais no qual se pode definir o valor percetível para o cliente. Estes pontos são demonstrados na Figura 9:

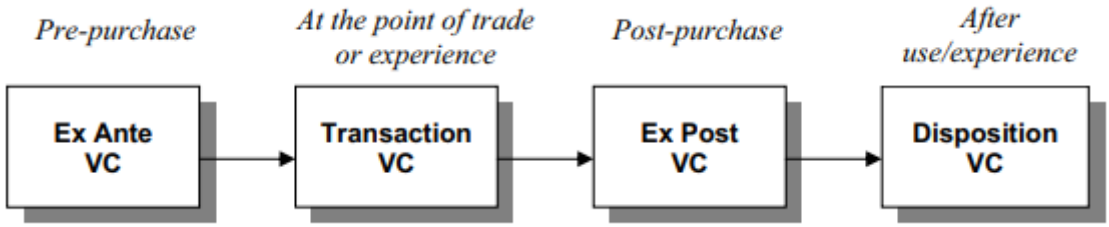


Figura 9 - Pontos Temporais Valor Perceptível

Seguindo estes pontos temporais, na Tabela 12 são definidos os benefícios e sacrifícios que o cliente pode obter nestes pontos:

Tabela 12 - Valor Percetível

Ponto	Benefício(s)	Sacrifício(s)
Ex Ante VC	- Melhoria do processo de suporte interno; - Melhor relação com os clientes; - Aumento da produtividade no departamento de suporte.	
Transaction VC	- Implantação e disponibilização do software; - Apoio na integração com a plataforma(s).	- Custo.
Ex Post VC	- Inovação do processo de suporte aos utilizadores; - Suporte técnico e lógico para ajuda na adaptação.	- Processo de Adaptação e transformação.
Disposition VC	- Melhoria na eficiência do departamento de help desk e consequente melhoria orçamental; - Melhoria nas relações com os clientes;	

	- Maior satisfação dos clientes; - Maior adesão às plataformas.	

3.3 Modelo Canvas

Criado pelo Suíço Alex Osterwalder, o modelo de canvas é um modelo que divide em 9 componentes as quatro áreas principais de um negócio: Clientes, Oferta, Infraestrutura e Viabilidade Financeira (Silva, 2020).

Desta forma é possível visualizar e definir facilmente os principais componentes de um produto. O modelo canvas para o serviço de help desk a desenvolver pode ser observado na Figura 1Figura 10, sendo que cada componente é posteriormente analisado nesta secção.

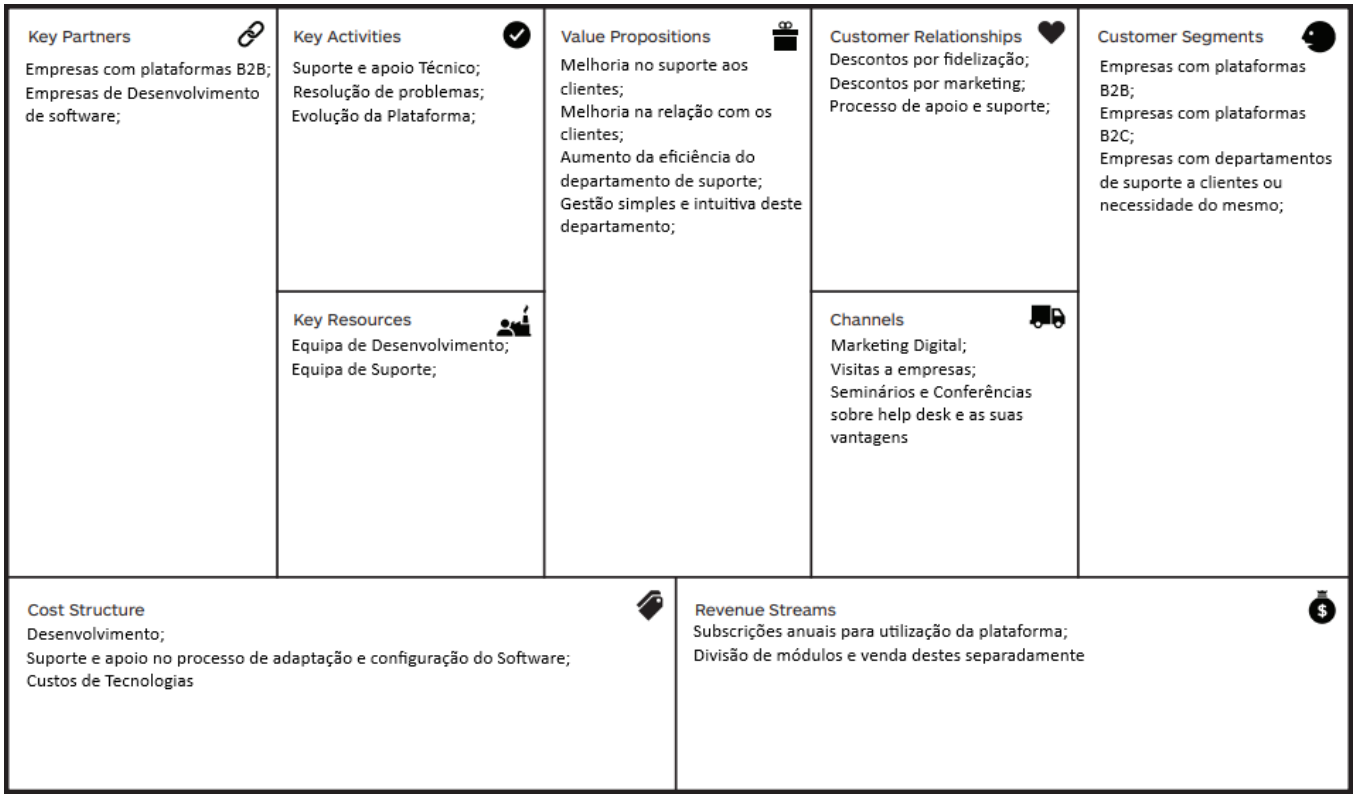


Figura 10- Modelo Canvas

Os segmentos de clientes (*Customer Segments*) representam os grupos de clientes a atingir. Como tal, englobam empresas com plataformas ou a utilizar ou a necessitarem de um serviço de help desk.

A proposta de valor (*Value Proposition*) representa o que o produto oferece aos clientes e as vantagens que estes podem obter ao aderir ao produto. Aqui podem ser incluídas algumas vantagens referidas ao longo da secção da Análise de Valor (cf. Secção 3), como melhoria no

suporte e relação com os clientes e uma melhoria nos processos e gestão interna do departamento de suporte (help desk).

Os canais de comunicação (*Channels*) representam a forma como o produto é divulgado e apresentado. Sendo assim, o principal canal será o marketing digital, através de publicidade online. No entanto, para divulgar um serviço de help desk, primeiramente, é necessário que as pessoas percebam a importância deste serviço, para o sucesso de uma organização. Sendo assim, também serão realizados seminários e conferências sobre o tema.

As relações com os clientes (*Customer Relationships*) representam os tipos de relacionamento estabelecidos com os mesmos, e o que faz com que estes se sintam ainda mais atraídos para o produto. Sendo assim, nesta secção podem ser incluídos diversos descontos feitos ao custo do produto, assim como o processo de suporte contínuo.

As fontes de receita (*Revenue Streams*) representam as formas como a empresa gera dinheiro. O produto será disponibilizado numa subscrição semestral/anual. Além da subscrição, o produto será dividido em módulos e para o acesso a alguns destes será necessário um pagamento adicional.

Os recursos chave (*Key Resources*) representam os recursos necessários para o funcionamento e sucesso do modelo de negócio. Como tal, pode-se incluir ambas as equipas de desenvolvimento e suporte, que permitem a evolução da plataforma e a satisfação e adaptação dos clientes.

As atividades chave (*Key Activites*) representam as atividades principais a realizar de forma constante, para garantir o fluxo do modelo de negócio. Nestas podem ser incluídas atividades de suporte e resolução de problemas dos clientes, assim como uma constante evolução da plataforma para se adaptar às necessidades do mercado.

As parcerias chave (*Key Partners*) representam os parceiros mais importantes para o funcionamento e sucesso do modelo de negócio. Neste caso, são as empresas que podem propagar o serviço desenvolvido, como: empresas com plataformas B2B, que o podem recomendar aos seus clientes e empresas de desenvolvimento de software, que o podem recomendar e integrar nas plataformas que desenvolvem.

A estrutura de Custos (*Costs Structure*) representa os custos necessários a este modelo de negócio. Estes são os custos de desenvolvimento (equipa e tecnologias) e os custos referentes ao processo pós-venda, processo de adaptação em que é necessário um apoio maior ao cliente. Design.

4 Análise e Design

Neste capítulo é realizada a análise do projeto a desenvolver, definindo os intervenientes e os requisitos que o sistema deve corresponder. Depois da análise ser feita, é desenhada a solução. Com este design pretende definir-se e planear os aspetos mais importantes do sistema e como estes devem ser implementados.

4.1 Requisitos

Nesta secção são descritos os requisitos do sistema, tanto os funcionais como os não funcionais. Nos requisitos funcionais, são descritos os casos de uso, ou seja, cada funcionalidade que o sistema fornece aos seus utilizadores. Enquanto que, nos requisitos não funcionais são levantados todos os pontos fundamentais ao sistema, mas que não constituem funcionalidades deste - como segurança e usabilidade. Previamente, são também apresentados os Atores do Sistema.

4.1.1 Atores

Os atores de um sistema são entidades que interagem com o sistema e que fazem uso das suas funcionalidades. Visto isto, foram identificados três atores do sistema:

- **Administrador:** Responsável pelo sistema e pelos colaboradores;
- **Colaborador:** Membro da equipa que oferece suporte aos utilizadores de uma aplicação/sistema;
- **Cliente:** Utilizador da aplicação/sistema que necessita de suporte.

4.1.2 Requisitos Funcionais

Nesta secção são identificados e descritos os casos de uso do sistema e como estes se relacionam com os utilizadores do sistema. Esta relação pode ser observada no diagrama de casos de uso da Figura 11 - Diagrama de casos de usoFigura 11, juntamente com os atores do sistema.

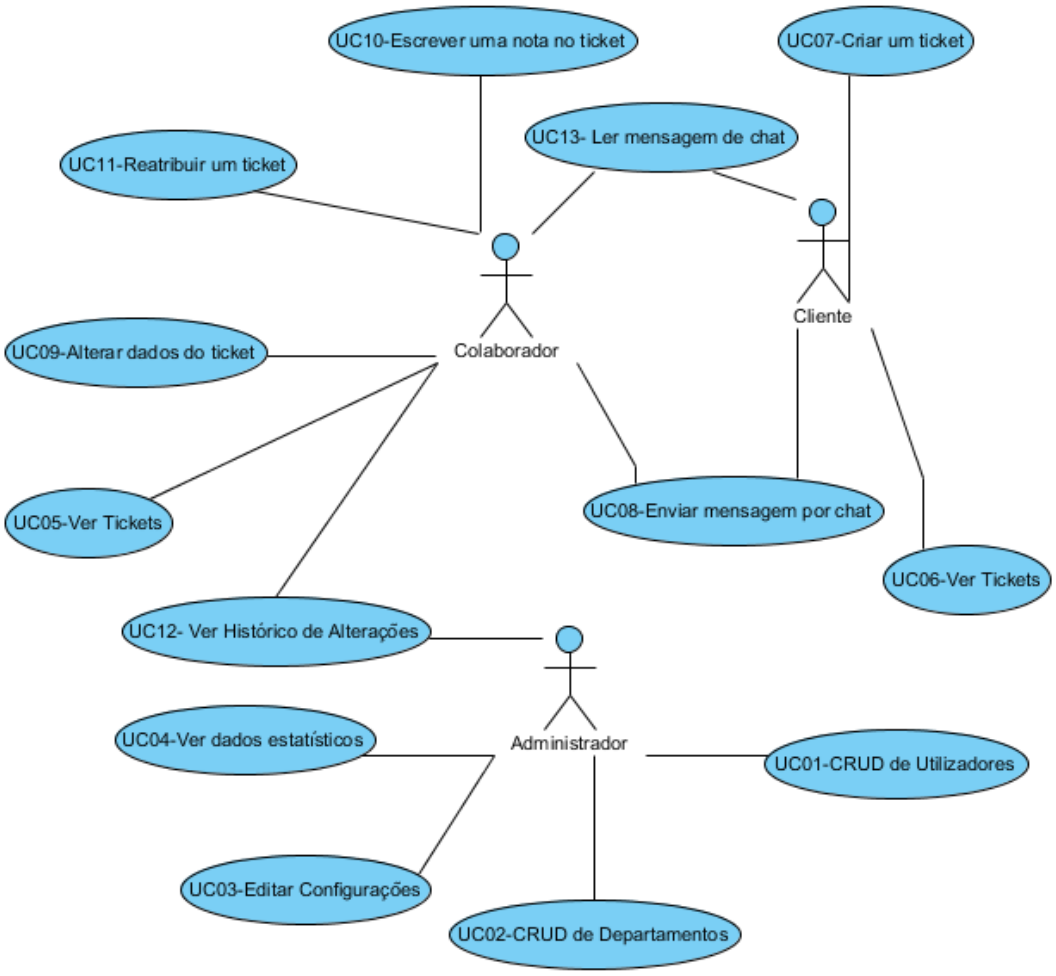


Figura 11 - Diagrama de casos de uso

De seguida, são brevemente descritos os casos de uso apresentados na Figura 11 - no que consistem e as condições necessárias para a sua realização.

4.1.2.1 UC01-CRUD de Utilizadores

CRUD (*Create Read Update Delete*) representa as ações de criar, ler, editar e apagar alguma entidade do sistema, neste caso, são utilizadores.

O Administrador deve ser capaz de criar utilizadores com acesso à plataforma, quer sejam novos Administradores ou Colaboradores, e posteriormente gerir estes.

Pós-Condições:

- Ação deve ser realizada e persistida com sucesso.

4.1.2.2 UC02-CRUD de Departamentos

O Administrador deve ser capaz de criar e gerir Departamentos. Um Departamento pode ter vários colaboradores e tickets associados. Esta associação será útil para o UC04-Ver Dados Estatísticos.

Pós-Condições:

- Ação deve ser realizada e persistida com sucesso.

4.1.2.3 UC03-Editar Configurações

O Administrador deve ser capaz de alterar as configurações da plataforma (ex: departamento a ser atribuído quando um ticket é criado).

Pós-Condições:

- Configurações devem ser gravadas com sucesso.

4.1.2.4 UC04-Ver Dados Estatísticos

O Administrador deve ter acesso a uma *dashboard* com dados estatísticos do sistema. Nesta *dashboard* devem existir os filtros relevantes.

Pós-Condições:

- Dados estatísticos são apresentados.

4.1.2.5 UC05-Ver Tickets (Cliente)

O Cliente deve ter uma plataforma própria, onde pode observar os detalhes de cada ticket criado por si.

Pós-Condições:

- Tickets são listados.

4.1.2.6 UC06-Ver Tickets (Colaborador/Administrador)

Na plataforma utilizada pelos Colaboradores e Administradores, estes devem conseguir observar os tickets existentes, podendo utilizar os filtros necessários. Os tickets devem aparecer numa listagem simples e concisa, onde são apresentados os campos mais relevantes. Após a seleção de um ticket, poderão observar os detalhes do mesmo.

É importante referir que um colaborador só pode ter acesso aos tickets atribuídos ao mesmo.

Pós-Condições:

- Tickets são listados.

4.1.2.7 UC07-Criar um ticket

O Cliente deve poder criar um ticket, podendo indicar o assunto do mesmo e uma breve descrição ou apenas iniciar uma conversa através do chat. Neste último caso, o ticket deve ser criado automaticamente pelo sistema.

Pós-Condições:

- Ticket é criado;
- Ticket é automaticamente atribuído a um colaborador e departamento;
- Colaborador atribuído recebe uma notificação sobre a atribuição;
- Evento de criação do ticket é gravado no histórico do mesmo.

4.1.2.8 UC08-Enviar mensagem por chat

O Cliente/Colaborador/Administrador, acedendo ao ticket, deve ser capaz de enviar uma mensagem através do chat, que será recebida em tempo real pelo outro Cliente/Colaborador.

Pré-Condições:

- Caso seja colaborador ou cliente, o ticket deve estar atribuído ao mesmo;
- Ticket deve estar criado.

Pós-Condições:

- Mensagem é enviada;

4.1.2.9 UC09-Alterar Dados de um Ticket

O Colaborador ou Administrador deve ser capaz de aceder a um ticket e alterar alguns dados, como por exemplo, o estado e prioridade do mesmo.

Pré-Condições:

- O Ticket deve estar criado;
- Caso seja colaborador, o ticket deve estar atribuído ao mesmo.

Pós-Condições:

- Mensagem é enviada;
- Evento da alteração é gravado no histórico do ticket.

4.1.2.10 UC10-Escrever notas no ticket

O Colaborador ou Administrador deve ser capaz de escrever notas num ticket para leitura interna. Estas notas não serão visíveis pelo cliente.

Pré-Condições:

- O Ticket deve estar criado;
- Caso seja colaborador, o ticket deve estar atribuído ao mesmo.

Pós-Condições:

- Nota fica gravada no ticket.

4.1.2.11 UC11-Reatribuir um ticket

Um colaborador ou administrador deve ser capaz de reatribuir um ticket a outro departamento ou colaborador. Caso seja apenas indicado o departamento, o sistema deve atribuir o ticket ao colaborador com menos tickets pendentes (tendo em consideração também a prioridade desses tickets).

Pré-Condições:

- O Ticket deve estar criado;
- Caso seja colaborador, o ticket deve estar atribuído ao mesmo.

Pós-Condições:

- Ticket é reatribuído;
- Evento da reatribuição é gravado no histórico do ticket.

4.1.2.12 UC12- Ver Histórico de Alterações

Acedendo a um ticket, o Colaborador ou Administrador deve ser capaz de observar o histórico de alterações realizadas a esse ticket, nomeadamente alterações ao estado, prioridade, categoria e colaborador. Este histórico deve informar a data, a alteração e o Utilizador que a realizou.

Pré-Condições:

- O Ticket deve estar criado;
- Caso seja colaborador, o ticket deve estar atribuído ao mesmo.

Pós-Condições:

- Histórico é apresentado.

4.1.2.13 UC13- Ler mensagem de chat

Após a mensagem ser enviada no UC08, esta deve ser obtida pelo recetor. Caso o chat esteja aberto, esta deve ser obtida em tempo real, no caso contrário, todas as novas mensagens devem ser obtidas quando a página do ticket é aberta.

Pré-Condições:

- O Ticket deve estar criado;
- Caso a página do ticket esteja aberta, mensagem deve ser obtida em tempo real.

Pós-Condições:

- Nova mensagem é apresentada.

Um dos requisitos pensados para esta solução era um *chatbot* (secção 2.5) - para estabelecer o primeiro contacto com o cliente. No entanto, devido a restrições de tempo, este requisito não será realizado nesta fase e será deixado para trabalho futuro.

4.1.3 Requisitos Não Funcionais

Nesta secção são apresentados os requisitos não funcionais do sistema, de acordo com o modelo FURPS+.

"(…) we can see that some requirements are functional and others nonfunctional, and also that some requirements are technology-independent and others technology-specific. And so we need a classification that will allow us to think about these different aspects of requirements. The template for the supplementary specifications artifact in RUP uses a classification that goes by the acronym FURPS+." (Eeles, 2004).

FURPS+ é um acrónimo que representa:

- *Funcionality* (Funcionalidades);
- *Usability* (Usabilidade);
- *Reliability* (Confiabilidade);
- *Performance* (Desempenho);
- *Supportability* (Suportabilidade);
- +: Requerimentos extra.

4.1.3.1 Funcionalidades

Aqui são apresentadas as funcionalidades não captadas nos casos de uso, pois não representam interações com utilizadores ou funcionalidades relevantes para o modelo de negócio.

- **Autenticação:** A utilização do sistema por parte dos três tipos de utilizadores requer uma autenticação por parte destes, através de um e-mail e uma palavra-passe. Também poderá existir autenticação por parte de serviços que também precisarão de uma identificação e palavra-chave.

- **Autorização:** Apesar de um utilizador estar autenticado, não implica que possa aceder a todos os recursos do sistema. Como tal, cada recurso tem de estar protegido para não ser disponibilizado aos atores errados (p.e: Um Colaborador não pode aceder a dados estatísticos, apenas os Administradores).
- **Segurança:** O sistema deve ser capaz de garantir a segurança e fiabilidade dos seus dados. Importante referir que também devem ser seguidas normas de confidencialidade como RGPD.
- **Integrações Externas:** O Sistema deve estar preparado para ser integrado por outros sistemas externos.
- **Logs:** Atividades efetuadas (especial menção para erros e possíveis falhas do sistema) devem ser registadas para poderem ser revistas posteriormente.

4.1.3.2 Usabilidade

A experiência do utilizador é algo bastante importante em cada sistema, como tal, nesta secção são levantados os requisitos que visão melhorar esta experiência e a usabilidade da plataforma.

- **Simplicidade:** As Plataformas não devem ser complicadas nem requerer competências especiais para serem utilizadas.
- **Fácil Utilização:** A interface das plataformas deve ser intuitiva e preparada para as funcionalidades mais utilizadas.
- **Elegante:** As interfaces gráficas da plataforma devem ser apelativas.

4.1.3.3 Confiabilidade

A Confiabilidade de um sistema representa a sua capacidade de evitar problemas e reagir em situações inesperadas.

- **Reação:** Caso um erro ou falha aconteça, o sistema deve ser capaz de o gerir, e corretamente e informar o utilizador do sucedido.
- **Tratamento:** Erros por parte do utilizador devem ser primeiramente evitados pelo sistema, no entanto, em segundo plano devem ser previstos e corrigidos ou o utilizador deve ser corretamente informado de como os solucionar.

4.1.3.4 Desempenho

O sistema deve ter uma boa performance e capacidade de resposta. À exceção das funcionalidades mais complexas ou pesadas, como o UC04-Ver Dados Estatísticos, o tempo de resposta para cada pedido deve ser inferior a 0.5 segundos.

4.1.3.5 Suportabilidade

Nesta secção são mencionados outros requisitos do sistema, como:

- **Testabilidade:** O Sistema deve possuir uma biblioteca extensa de testes para garantir a sua Confiabilidade.
- **Manutenção:** O Sistema deve estar preparado para alterações futuras e devem ser adotados padrões de desenvolvimento que suportem e facilitem estas alterações.
- **Escalabilidade:** O Sistema deve estar preparado para novos requisitos e funcionalidades, e devem ser adotados padrões de desenvolvimento que os suportem e facilitem.

4.1.3.6 Requerimentos extra

Nesta secção são descritos outros requerimentos que não se insiram nas restantes secções do FURPS+, como restrições de design, restrições de implementação, etc.

- **Boas práticas de desenvolvimento e design:** Durante o processo de Design e Desenvolvimento do sistema, devem ser adotadas boas práticas e padrões que assegurem a confiabilidade e qualidade do sistema.
- **Cobertura de Código nos testes Unitários:** É estipulado que os testes unitários possuam um mínimo de 85% de cobertura do código.

4.2 Arquitetura

"When people in the software industry talk about "architecture", they refer to a hazily defined notion of the most important aspects of the internal design of a software system. A good architecture is important, otherwise it becomes slower and more expensive to add new capabilities in the future." (Fowler, 2019).

A arquitetura a utilizar no sistema é uma das decisões chave a tomar antes do desenvolvimento do mesmo. Devem ser identificados todos os componentes que compõem o sistema e como estes interagem uns com os outros.

Existem vários tipos de arquitetura (secção 2.4.1), em seguida serão apresentadas as alternativas principais para a arquitetura do sistema a desenvolver, serão analisadas e no final será identificada e detalhada a escolhida.

4.2.1 Análise de Possíveis Arquiteturas

Para cada alternativa será apresentado um diagrama de componentes, que identifica os componentes principais dessa alternativa e como estes comunicam entre eles. Em seguimento da análise feita na secção 2.4.1, existem 2 alternativas a analisar: uma monolítica e outra baseada em micro serviços.

4.2.1.1 Arquitetura monolítica

O tipo de arquitetura monolítico é caracterizado por conter apenas um serviço em *back end*, que contém toda a lógica de negócio e persistência de dados. É possível analisar o diagrama de componentes para esta alternativa, na Figura 12.

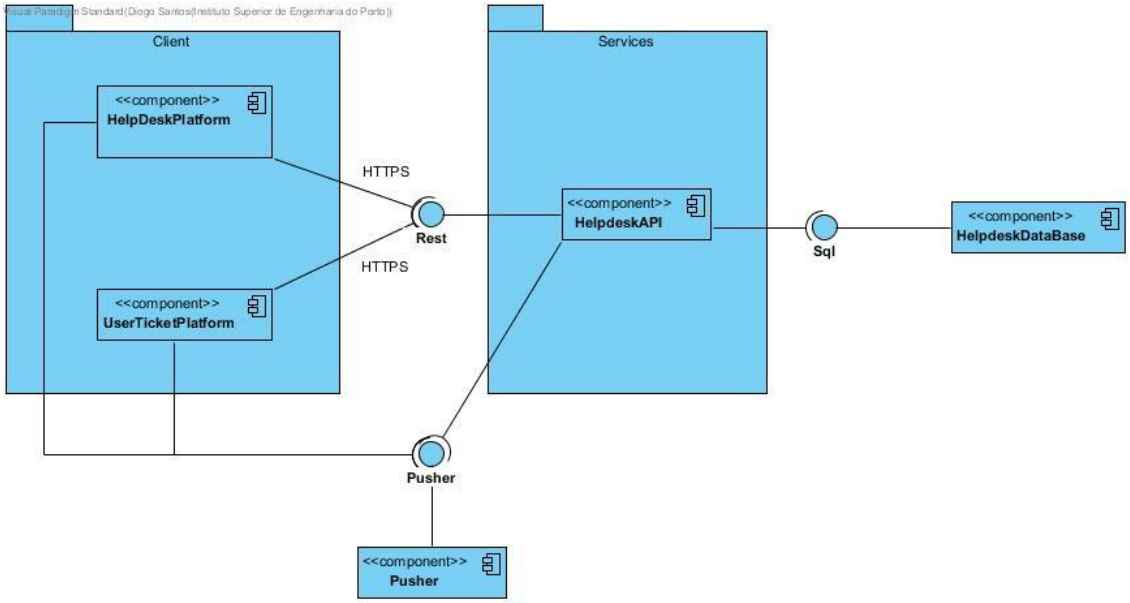


Figura 12- Diagrama de Componentes da Arquitetura Monolítica

Relativamente á área do cliente, ou seja, aos componentes da aplicação disponíveis para os clientes e com uma interface visual, existem 2 plataformas:

- **UserTicketPlatform-** Plataforma utilizada pelos utilizadores, que permite a criação e seguimento de tickets. Corre num web browser e pode ser acedida por qualquer utilizador, desde que este tenha algum tipo de autenticação na plataforma que está a utilizar.
- **HelpDeskPlatform-** Plataforma utilizada pelo departamento de *help desk*. Corre num web browser e pode ser acedida por qualquer colaborador registado.

Relativamente aos serviços da plataforma, existe apenas um serviço responsável por toda a lógica de negócio do sistema: o "HelpDeskAPI". Este serviço pode ser acedido por ambas as plataformas, através do protocolo de HTTPS, seguindo as várias regras e restrições impostas pelo padrão REST.

Nesta arquitetura, a melhor estrutura de base de dados a adotar seria uma base de dados única e relacional, como é costume com este tipo de arquitetura, uma das possíveis linguagens na qual definir esta base de dados seria SQL.

No diagrama da Figura 12 também se pode observar um componente "Pusher". Este componente representa o cliente adotado para o chat em tempo real. Como tal, é acedido por ambas as plataformas para ler as mensagens e pelo serviço para as escrever. O funcionamento deste componente vai ser explicado de forma mais detalhada na secção 4.3 e no capítulo 5.

4.2.1.2 Arquitetura de Micro Serviços

A arquitetura de micro serviços carateriza-se pela divisão em pequenos serviços, sendo que, cada um é desacoplado dos outros e tem uma responsabilidade no sistema. Na Figura 13 é apresentado o diagrama de componentes para esta alternativa:

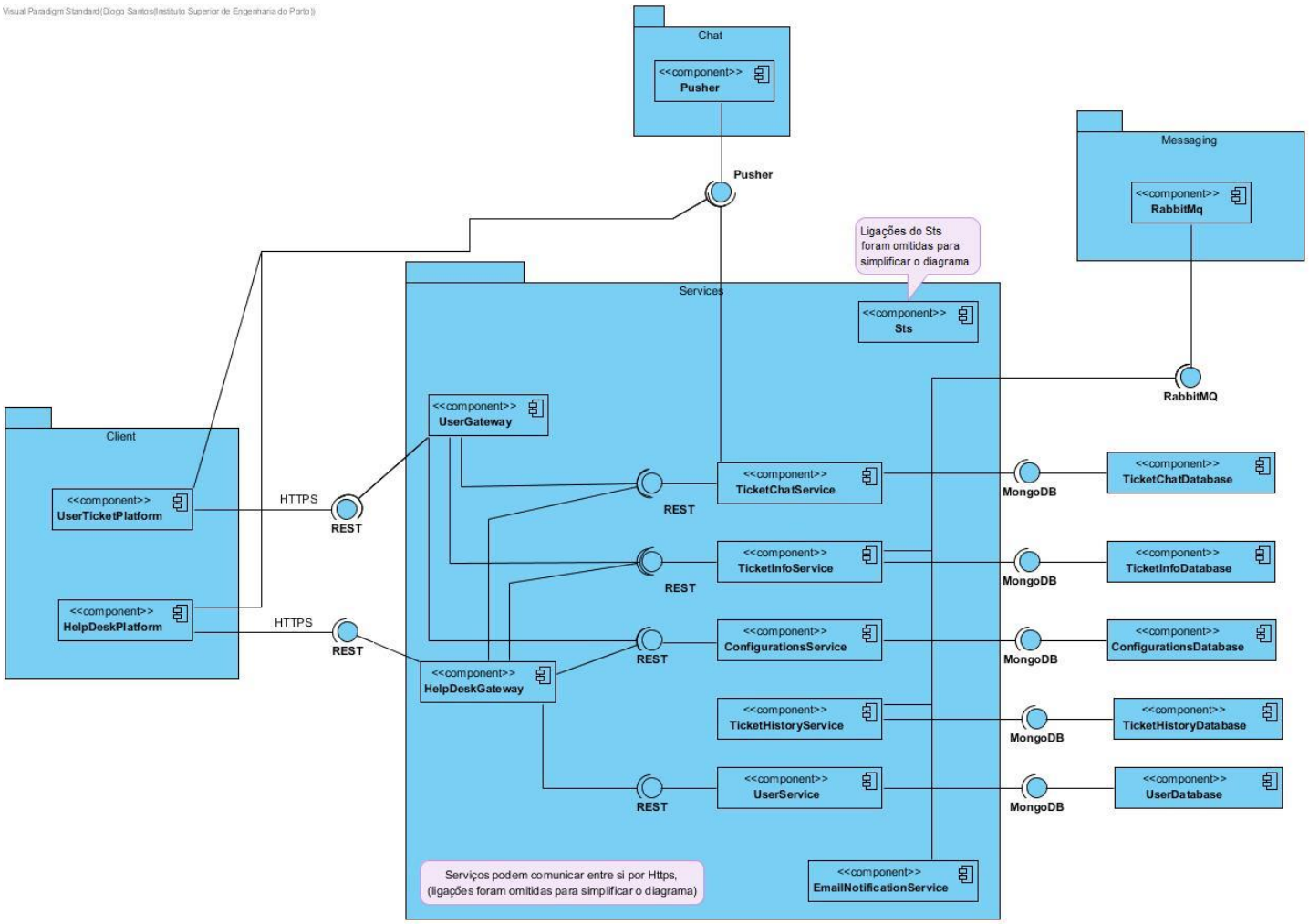


Figura 13- Diagrama de Componentes da Alternativa de Micro Serviços

Como se pode observar, os componentes a nível do cliente, mantêm-se. É nos serviços e na sua comunicação que a complexidade aumenta. Existem dois Gateways, que são os componentes responsáveis por intermediar a comunicação entre o *Front End*(Cliente) e o *Back End*(Serviços), sendo que ambos têm de estar desacoplados. A utilização de *gateways* evita o *Front End* de ter de "conhecer" a estrutura do Back End. Desta forma, existe um *gateway* para cada plataforma. É importante referir que os *gateways* apenas funcionam como intermediários e não como orquestradores de forma alguma.

Na camada de serviços existem os micro serviços que constituem o sistema; cada um com a sua responsabilidade.

Outro padrão e boa prática desta arquitetura é o *Database per Service*, que define que cada micro serviço deve ter a sua própria base de dados independente. Aproveitando o facto de os serviços serem desacoplados e independentes, assim como no intuito de assegurar uma melhor performance, é costume a utilização de bases de dados não relacionais, como por exemplo "MongoDB".

Também é possível observar a utilização do padrão arquitetural de *Event Sourcing,* aplicado através do "RabbitMQ". Através deste componente, é possível os serviços publicarem e subscreverem a eventos. Este padrão é descrito com maior detalhe na secção 2.4.2.

Tal como na alternativa de arquitetura monolítica, é usado um cliente de chat - o "Pusher". Para ser possível o chat em tempo real, entre ambas as plataformas, o funcionamento deste componente vai ser explicado de forma mais detalhada nas secções 4.3 e 5.

4.2.1.3 Decisão e Detalhes da Arquitetura a adotar

Existem várias vantagens entre cada uma das arquiteturas (cf. Secção 2.4.1.4), sendo que a maior diferença a destacar é a baixa complexidade e facilidade de desenvolvimento inicial da arquitetura monolítica. No entanto, esta facilidade apenas se verifica no desenvolvimento inicial, porque quando é preciso escalar o produto e desenvolver novas funcionalidades já se torna mais complicado e desvantajoso. Mesmo a baixa complexidade só se verifica num nível arquitetural. Sendo responsável por todo o negócio do sistema, o serviço monolítico vai ser bastante complexo e para cada nova funcionalidade terá de ser alterado.

Sendo assim, verifica-se que a arquitetura de micro serviços será a ideal para este produto, pois permite uma escalabilidade mais simples, tornando possíveis novas funcionalidades serem facilmente implementadas sem comprometer o resto do sistema.

Voltando a analisar a Figura 13, verifica-se que existem os seguintes micro serviços:

- **TicketInfoService-** Serviço responsável pela informação principal de um serviço;
- **TicketChatService-** Serviço responsável pelo chat entre os utilizadores e colaboradores;
- **TicketHistoryService-** Serviço responsável pelo armazenamento e disponibilização do histórico de ações de um ticket;
- **UserService-** Serviço responsável pela gestão de utilizadores (Colaboradores e Administradores);
- **ConfigurationsService-** Serviço responsável pelas configurações do sistema, incluindo os Departamentos;
- **EmailNotificationService-** Serviço responsável pelo envio de e-mails;
- **Sts (Security Token Service)-** Serviço responsável pelo autenticação e autorização, será mais detalhado nas secções 4.3.2 e 5.1.

Cada serviço (com a exceção do EmailNotificationService que não tem ligação a uma base de dados) será desenvolvido, seguindo uma arquitetura de 3 camadas. Esta arquitetura define a divisão do serviço em 3 camadas: Apresentação, Lógica e Persistência de Dados. Na Figura 14 está representado um diagrama de componentes para o "TicketInfoService", que representa as 3 camadas:

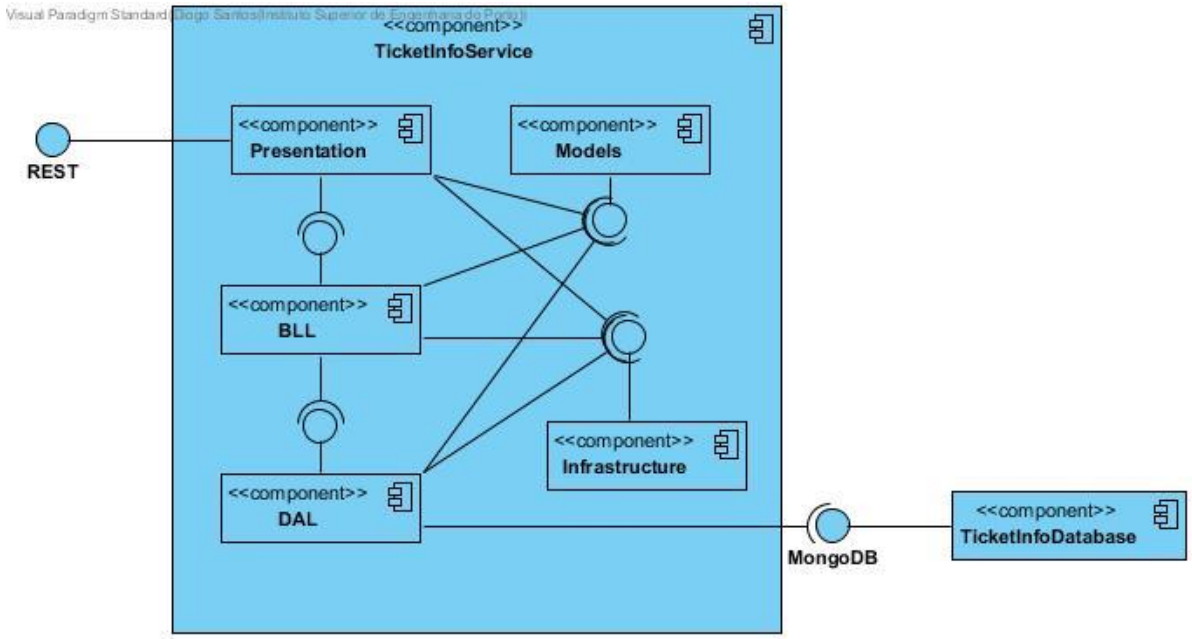


Figura 14- Diagrama de componentes do "TicketInfoService"

Como se pode observar pela Figura 14, existem 5 componentes principais no serviço, sendo que 3 deles representam cada uma das camadas. Também se pode observar que o serviço é exposto pelo componente "Presentation" e que apenas o componente "DAL" tem acesso à base de dados. Segue uma análise detalhada de cada uma das camadas:

- **Apresentação-** Representada na Figura 14 pelo componente "Presentation", esta camada é composta pelos controllers e lógica de API do serviço, sendo que é nesta camada que é definida a interface de exposição do serviço. Devem ser adotados e seguidos padrões e boas práticas de desenvolvimento como "REST" e *Depency Injection*.
- **Lógica-** Representada pelo componente "BLL" (*Business Logic Layer*) e sendo acedida apenas pela API, esta camada é responsável pela lógica de negócio do serviço. Devem ser adotados e seguidos padrões e boas práticas de desenvolvimento como "SOLID" e "GRASP" e "Dependency Injection".
- **Persistência de dados-** Representada pelo componente "DAL" (*Data Access Layer*), esta camada é responsável pela persistência de dados e conexão com a base de dados. Devem ser adotados e seguidos padrões e boas práticas de desenvolvimento como o padrão Repositório e "Dependency Injection".
- **Infraestrutura:** Representado pelo componente "Infrastructure", este componente é responsável por todas as ações independentes de uma camada em si e, como tal, pode ser acedido por todas as camadas. Alguns exemplos do que se pode inserir neste componente seriam: Comparações de Datas; Classes de Configuração.
- **Modelos:** Representado pelo componente "Models", representa todos os modelos do serviço, como modelos de domínio, "DTOs" e "ViewModels".

4.3 Design

O objetivo desta secção é planear e desenhar a solução. Primeiramente é definido o modelo de domínio, seguido pelo processo de autenticação e autorização onde é demonstrado o funcionamento do IdentityServer4 e como este se enquadra na solução a implementar. De seguida, são desenhados os casos de uso mais complexos da solução e é detalhado o fluxo destes e todas as interações necessárias.

4.3.1 Diagrama de Classes

O diagrama de classes representa visualmente as classes que constituem o sistema, com as relações entre cada uma. Com este modelo é possível ter uma visão global dos conceitos do sistema e de como estes se relacionam, e assim proporcionar uma boa visão do domínio da solução.

É importante referir que, estando perante uma arquitetura de micro serviços, estas classes estão distribuídas entre os vários serviços. As classes estão relacionadas conceitualmente, no entanto, como foi adotado um sistema de base de dados não relacional, estas ligações não vão estar implementadas nas próprias bases de dados.

Este diagrama está representado na Figura 15:

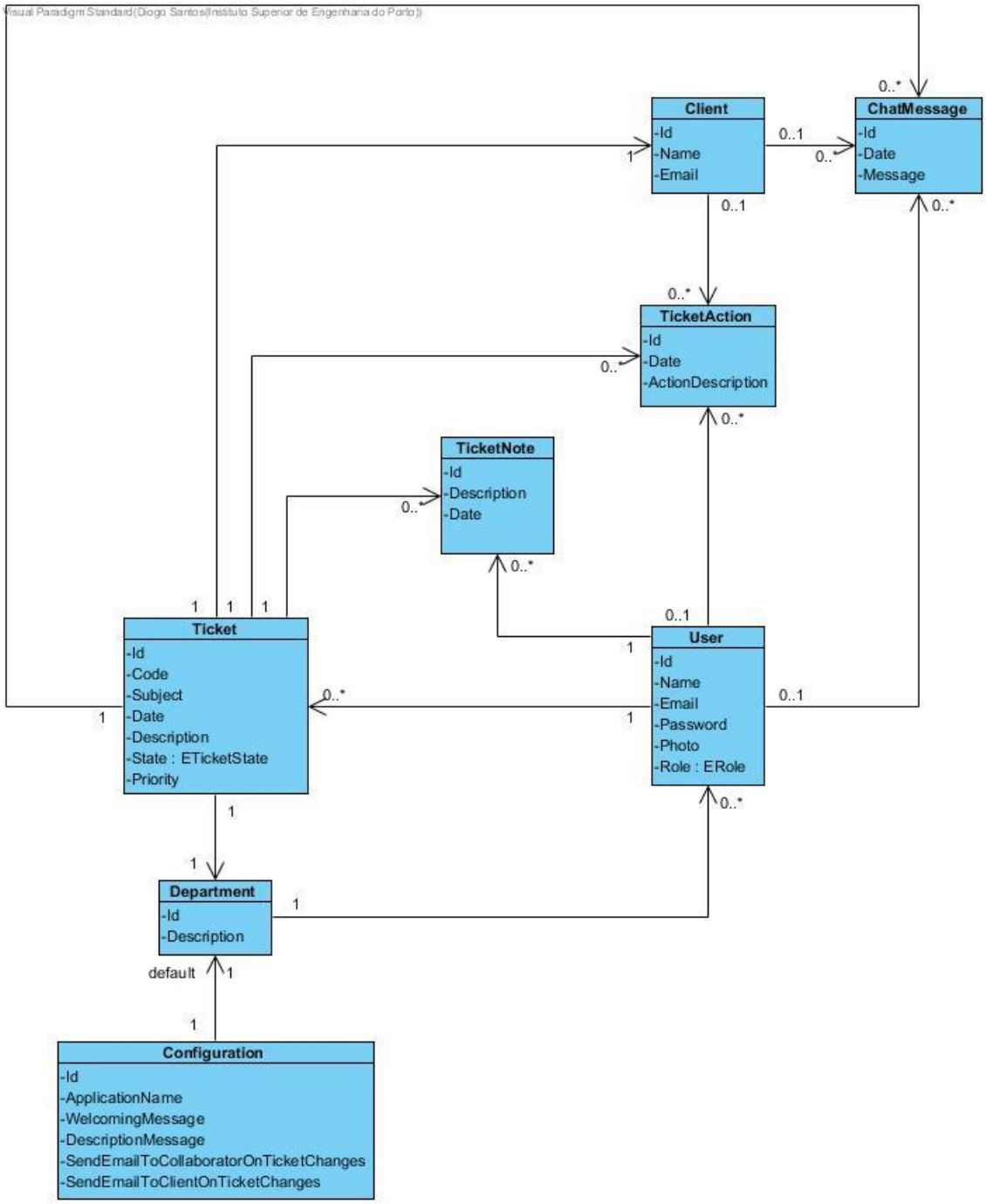


Figura 15 – Diagrama de Classes

Como é possível observar, existem várias classes e na implementação estas são divididas entre os vários serviços. O diagrama da Figura 15 representa estas classes, os seus atributos e as suas relações, no entanto, existem alguns itens que vale a pena especificar, como:

- "Configuration": Classe que define as configurações do sistema - esta classe possui diversos atributos relevantes para o sistema como:
    - Departamento *default*: Departamento para o qual todos os tickets vão ser atribuídos quando são criados;

    - "ApplicationName": Nome a aparecer na "ClientTicketPlatform";
    - "WelcomingMessage": Mensagem a aparecer como título na *dashboard* da "ClientTicketPlatform";
    - "DescriptionMessage": Mensagem a aparecer como subtítulo da *dashboard* da "ClientTicketPlatform";
    - Os restantes atributos são campos que definem se se deve ou não enviar e-mail quando são realizadas alterações no "Ticket".
- "ChatMessage" e "TicketAction": É importante referir que ambas as classes podem ter um cliente ou um colaborador ("User") definidos. No entanto, é obrigatório ter um destes.
- "User": O Campo "Role" é um *enum* que pode ser definido como Administrador ou Colaborador, aquando da criação do mesmo.
- "Ticket": Sendo a classe mais importante para o sistema é importante definir todos os seus atributos:
    - "Code": Código do Ticket, a ser automaticamente gerado pelo sistema quando o Ticket é criado;
    - "Subject": Assunto do Ticket;
    - "Description": Descrição do Ticket, no caso de o ticket ser criado através de uma mensagem, esta deve ser igual à mensagem que o criou (até ser eventualmente alterada).
    - "Date": Data de Criação do Ticket
    - "State": *Enum* que representa o estado do Ticket, podendo adotar as seguintes alternativas:
        - Criado;
        - Atribuído;
        - Em Tratamento;
        - Fechado.
    - "Priority": Representa a prioridade do Ticket - pode variar entre 1 e 5 (o 5 é o valor máximo recomendado, mas é possível definir um mais alto) - sendo que o 1 é a prioridade máxima.

Um Ticket pode ter várias mensagens, várias notas, vários registos de histórico e tem de ter um colaborador, um cliente e um departamento.

4.3.2 Autenticação e Autorização

Estando perante um sistema de micro serviços, garantir a autenticação e autorização torna-se um desafio maior. É necessário garantir que todos os acessos aos serviços são autorizados e que existe apenas um processo de autenticação. Também é necessário que cada serviço seja capaz de fazer pedidos HTTP autenticados a outros serviços. São estes requisitos que o "Identity Server" vem resolver. O "Identity Server" é uma *framework* para ".net core", baseada em "OpenId Connect" e "OAuth2.0".

Esta “framework” tem várias funcionalidades, sendo que irão ser detalhadas somente as relevantes para a solução a desenvolver. No entanto, é importante primeiro distinguir Autenticação de Autorização:

- **Autenticação:** Normalmente definido como *Log In*, a autenticação é o processo em que uma entidade é validada consoante um conjunto de credenciais e caso seja válido, é dado o acesso a um recurso ou sistema.
- **Autorização:** A Autorização é o processo de definir se uma entidade tem permissões para aceder a um recurso.

O “Identity Server” oferece a capacidade de uma única entidade ser autenticada e autorizada, num conjunto de serviços ou recursos. Para isto, é necessária a existência de um serviço específico denominado “Security Token Service” (STS). Este serviço é responsável por:

- Autenticação;
- Geração de Token (Conjunto de caracteres encriptados que são usados para provar a autenticação);
- Validação de Token;
- Outros (Não relevantes para o sistema a desenvolver).

Tendo isto em conta, o fluxo de autenticação simplificado é o seguinte:

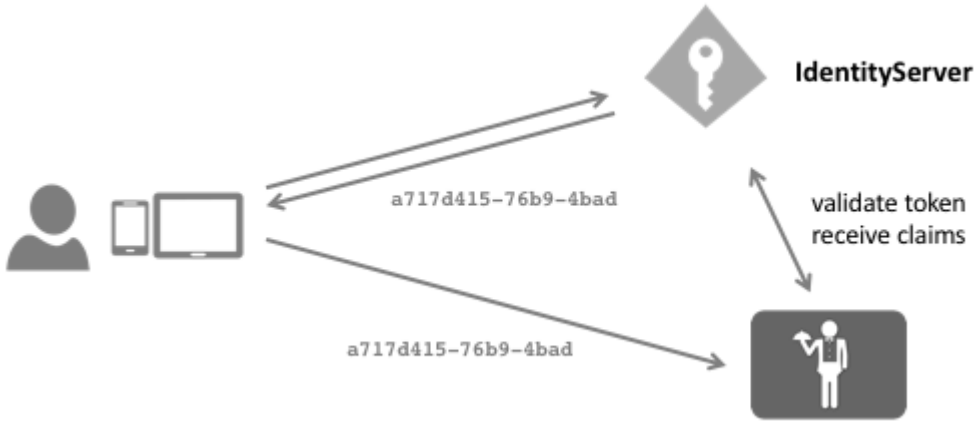


Figura 16- Fluxo de Autenticação “Identity Server”

Como é possível observar na Figura 16, retirada de (IdentityServer4, 2016), o fluxo começa por um utilizador (ou um serviço) fazer um pedido ao “identity server”, para obter o token (/connect/token). Caso o pedido seja válido, o “identity server” retorna o token gerado. Após receber o token, o utilizador usa o mesmo para fazer um pedido a um outro serviço. Este serviço faz outro pedido ao STS, para validar o token recebido. Caso seja válido, é porque o utilizador está autenticado.

O STS aceita vários fluxos de autenticação, no entanto, para o sistema a desenvolver, são apenas necessários 2 destes:

- Credenciais de Cliente: utilizado para autenticar serviços;
- Password: Utilizado para autenticar utilizadores.

Para o fluxo de Credenciais de Cliente, são necessários os seguintes campos:

- "Client_Id": Identificador do cliente;
- "Client_Secret": Chave do cliente;
- "Scopes": Recursos aos quais esse cliente requer acesso.

Os "scopes" são recursos predefinidos para o cliente nas configurações do STS. Estes recursos são usados para a autorização. É possível, por exemplo, definir que um certo cliente só tem acesso aos recursos *ticket* e *departments*, caso o cliente peça o recurso *configurations* ao STS, este vai retornar erro pois o cliente não tem este acesso configurado. Caso um cliente tente aceder a um recurso sem o scope necessário, este vai retornar o código 403 para indicar que o cliente não tem autorização.

O fluxo de password pode ser mais complexo utilizando OAuth2.0, este está preparado para utilizadores públicos, no entanto, visto que o acesso ao sistema é realizado por utilizadores e administradores, alguns passos não são relevantes. É importante também referir que os utilizadores fazem parte de um cliente e é o próprio cliente que autentica esse utilizador, por isso, os dados do cliente são também necessários. Sendo assim, o pedido requer os seguintes campos:

- "Client_Id": Identificador do cliente;
- "Client_Secret": Chave do cliente;
- "Scopes": Recursos aos quais esse cliente requer acesso;
- "Username": Identificador do utilizador;
- "Password": Chave do utilizador.

Tendo estes fluxos, é agora possível definir os clientes a criar no STS, que serão:

- "client_ticket_application": Cliente para a plataforma à qual os clientes acedem (Fluxo de credenciais de cliente);
- "ticket_application": Cliente para a plataforma de help desk (Fluxo de password);
- "ticket_info_service": Cliente para o serviço "TicketInfoService" (Fluxo de credenciais de cliente).

O Cliente através do qual os administradores e colaboradores se vão autenticar é o "ticket_application".

Visto que o "ticket_info_service" irá precisar de realizar alguns pedidos a outros serviços, vai também precisar de um cliente para se autenticar.

Passando para a autorização - são necessários doistipos diferentes de autorização, no sistema a desenvolver.

- **Autorização por recurso**: Todos os clientes e utilizadores com o scope correto são autorizados a aceder a esse recurso. Ex: Um cliente com o scope de *departments* está autorizado a aceder aos departamentos;

- **Autorização por *Claim***: Tal como os clientes têm acesso a scopes, os utilizadores têm acesso a outras configurações definidas por *claims* (podem ser usadas para guardar o id e nome do utilizador por exemplo). Uma das *claims* necessárias para o sistema a desenvolver é a do Cargo do utilizador (Administrador ou Colaborador), visto que existem recursos aos quais apenas os administradores podem aceder. Estes recursos têm de ser protegidos pela *Claim* de cargo, certificando-se que o valor desta é administrador. Ex: Apenas um administrador pode criar, editar ou apagar departamentos.

É importante referir que um recurso pode ter de ser protegido por scope e por cargo. Utilizando o exemplo dos departamentos, um serviço com o scope de *departments* tem de poder criar departamentos, no entanto, caso seja um utilizador, este tem de ter o cargo de administrador.

O processo e restrições de autorização vão ser detalhados na secção 5.1.

4.3.3 Casos de Uso

Nesta secção será feito o *design* dos casos de uso mais complexos ou relevantes para o sistema. O objetivo é que a análise feita destes casos de uso seja suficiente para perceber o funcionamento principal dos restantes.

4.3.3.1 UC07- Criar um ticket

Sendo um dos casos de uso mais essenciais para o sistema, é também um dos mais complexos. O Caso de uso começa pelo acesso à plataforma de tickets por parte do cliente. Este pode ser feito através de um pedido para poder ser ativado em qualquer outra plataforma ou serviço. Nesse pedido devem constar as informações (identificador, nome e e-mail) do utilizador que o está a fazer, para poderem ser gravadas e utilizadas nas restantes ações.

Quando o utilizador tem acesso à plataforma, tem a opção de criar um ticket. Após esta seleção, é redirecionado para a página de criação do ticket, onde é pedido que insira as informações necessárias. Depois deste processo e da seleção da opção para criar o ticket, é feito um pedido para o sistema, que inicia a sua criação. No Diagrama de Sequência da Figura 17 é possível observar este fluxo, assim como a interação de todos os componentes necessários.

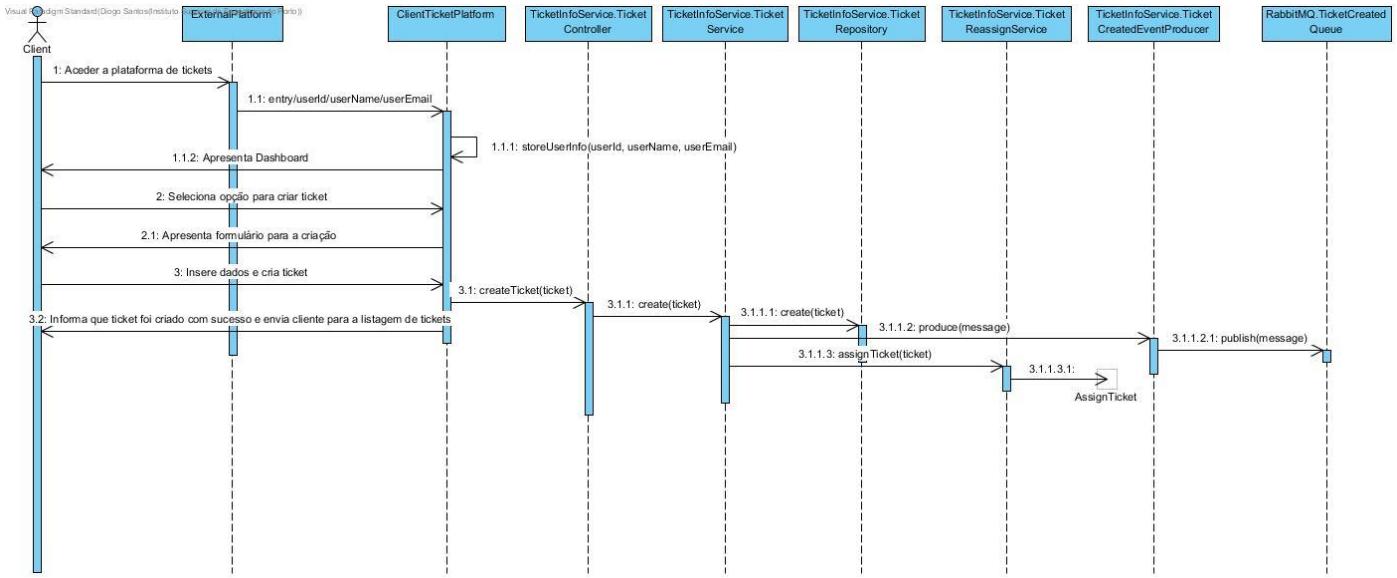


Figura 17 - Diagrama de Sequência UC07

Como é possível observar na Figura 17, existem várias ações aquando da criação do ticket. É importante referir que, após o ticket ser criado no repositório, todas as ações que seguem são assíncronas - o que significa que são executadas em paralelo. Isto acontece para garantir o desacoplamento destas ações (Atribuição de Colaborador ao ticket e publicação da mensagem de ticket criado). Desta forma, mal o ticket é criado no repositório - e consequentemente na base de dados -, é retornada uma mensagem de sucesso para o Cliente. É possível também observar que é publicada uma mensagem no *message broker* adotado, seguindo o fluxo, esta mensagem será posteriormente lida pelo "TicketHistoryService" e será adicionado um registo de criação do ticket á data atual.

Para simplificar o diagrama de sequência da Figura 17, o fluxo de atribuição do ticket a um colaborador (assinalado por "AssignTicket") pode ser observado na Figura 18.

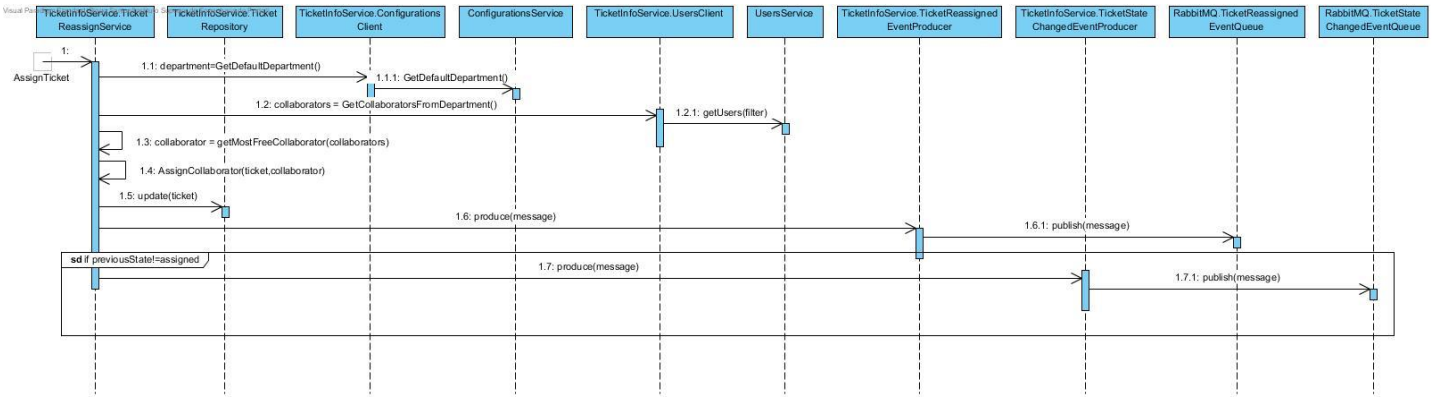


Figura 18 - Diagrama de Sequência Atribuir Ticket

Como é possível observar na Figura 18, para atribuir um ticket é necessário obter o colaborador mais adequado para esse ticket. Para isso, é preciso obter o departamento definido como ponto

de entrada para o ticket (este departamento pode ser definido nas configurações) e depois obter todos os colaboradores desse departamento. Após isto, é obtido o colaborador que está mais disponível, através de um algoritmo que classifica a disponibilidade dos colaboradores, de acordo com o número de serviços abertos e a sua prioridade.

Depois de ser obtido o colaborador, atribuímo-lo ao ticket, alterando o também o seu estado para "Atribuído". Após este passo, basta apenas atualizar o ticket na base de dados.

Também é possível notar que são gerados dois eventos (novamente assíncronos) neste fluxo. Ambos serão consumidos pelo "TicketHistoryService" para gerar registos de alterações ao ticket, o evento de alteração de colaborador também poderá ser lido pelo "EmailNotificationsService" para enviar uma notificação ao colaborador atribuído.

É importante referir que este fluxo também é utilizado pelo UC11-Reatribuir um ticket, sendo que, caso seja definido o colaborador a atribuir, o fluxo inicia no passo 1.4 do Diagrama da Figura 18. Caso seja apenas definido o Departamento, este inicia no passo 1.2.

Este caso de uso tem outro fluxo, que é criar o ticket a partir de uma mensagem. O objetivo é, caso o utilizador queira apenas enviar uma mensagem com a pergunta, seja mais simples. Neste caso, o utilizador apenas tem de escrever uma mensagem e o ticket é gerado em *Back End*. Na figura Figura 19 é possível observar este fluxo:

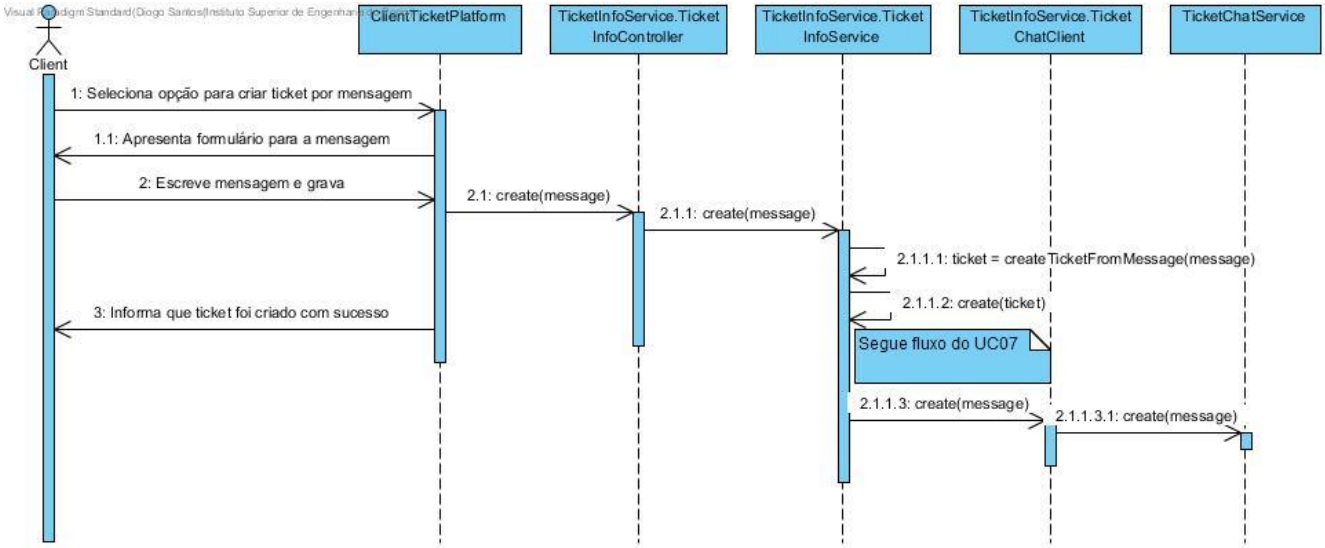


Figura 19- Diagrama de Sequência de criação de ticket a partir de mensagem

Como se pode observar, o fluxo é bastante semelhante ao de criar ticket. Primeiramente é gerado o ticket a partir da mensagem, com um assunto e descrição criados a partir da mesma. Depois de criar o ticket, da mesma forma como é criado na Figura 17, é feito um pedido HTTP para o "TicketChatService", para guardar a mensagem.

É pretendido que depois de ler a mensagem, o colaborador altere o assunto e a descrição do ticket, de acordo com a mesma.

4.3.3.2 UC08- Enviar mensagem por chat

Uma das funcionalidades principais deste sistema é o chat entre o cliente e o colaborador. Como tal, este é um dos casos de uso mais importantes e um dos mais complexos.

Para o chat é necessário um cliente próprio, algo onde seja possível publicar e subscrever ás mensagens, o cliente a adotar é o "Pusher", este cliente foi adotado pois tem bastante documentação sobre integração com "Angular" e ".Net Core". O "Pusher" funciona por subscrição de mensagens. A implementação é idêntica à do "RabbitMQ", apesar do funcionamento não ser exatamente o mesmo. A responsabilidade de escrever a mensagem é do "TicketChatService", que deve escrever a mensagem depois de a guardar na base de dados. Ambas as plataformas de *Front End* subscrevem o "Pusher" de forma que, mal uma mensagem seja escrita, é lida pela plataforma. Os detalhes de implementação vão ser mais detalhados na secção 5.4.2.

O fluxo deste caso de uso pode ser observado no diagrama de sequência da Figura 20.

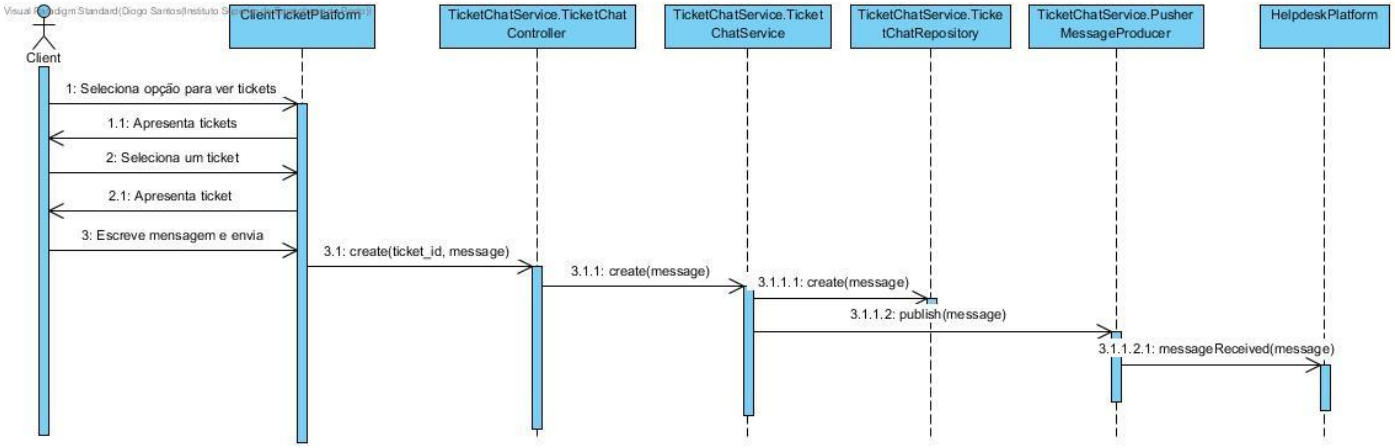


Figura 20- Diagrama de Sequência UC08

O diagrama representado na Figura 20 parte do pressuposto que existe um colaborador com o ticket aberto, dessa forma, a plataforma está a subscrever a fila de mensagens daquele ticket. Caso isso não aconteça, quando o ticket é aberto pelo utilizador a mensagem é obtida através de um pedido "Get", feito ao "TicketChatService".

Como se pode observar no diagrama da Figura 20,o chat está presente dentro da página do ticket. Neste chat são apresentadas as mensagens todas até ao momento. Após o utilizador escrever e enviar a mensagem, esta é criada no "TicketChatService". Após a mensagem ser criada na base de dados com sucesso, é publicada no "Pusher" para poder ser lida caso haja subscritores para aquele ticket. No caso de haver subscritores, a mensagem é recebida e apresentada.

Acontece o mesmo para a mensagem aparecer no chat do Cliente - esta aparece como subscrição do "Pusher" - o que significa que, caso haja algum problema (como o "Pusher" estar

em baixo ou a mensagem não ser gravada com sucesso) a mensagem não irá aparecer ao Cliente depois deste a enviar.

4.3.4 Pipelines (CI/CD)

Continuous Integration (CI) e *Continuous Delivery* (CD) são 2 processos bastante importantes, porque permitem automatizar o processo de testes e publicação dos serviços.

A responsabilidade do CI é correr os diversos testes da aplicação, cada vez que haja uma alteração e, dessa forma, garantir que tudo continua estável após a alteração. A CI deve funcionar da seguinte forma: quando é feita uma alteração à aplicação e o código é enviado para o repositório, deve ser iniciada uma pipeline de CI para aquela versão. Esta pipeline deve correr os diversos testes que possam existir naquele serviço e retornar se estes passaram todos ou não. Desta forma, é possível a equipa de desenvolvimento perceber imediatamente se algo está errado com as últimas alterações e, caso esteja, corrigir.

A pipeline de CI deve correr aquando de qualquer alteração em qualquer *branch*.

A responsabilidade de CD é diferente. Esta pipeline é responsável por publicar o serviço. Desta forma, após a pipeline de CI correr com sucesso, é iniciada a pipeline de CD que publica as alterações efetuadas automaticamente. É importante referir que a pipeline de CD só deve correr no *branch* de “master” e que, como tal, as alterações só devem ser submetidas neste *branch* quando o objetivo for que sejam publicadas.

4.3.5 Diagrama de Implantação

O Diagrama de implantação representa como o sistema deve ser implantado. Representa nomeadamente o hardware onde cada componente deve ser publicado e os protocolos de comunicação entre cada componente. Este diagrama está representado na Figura 21.

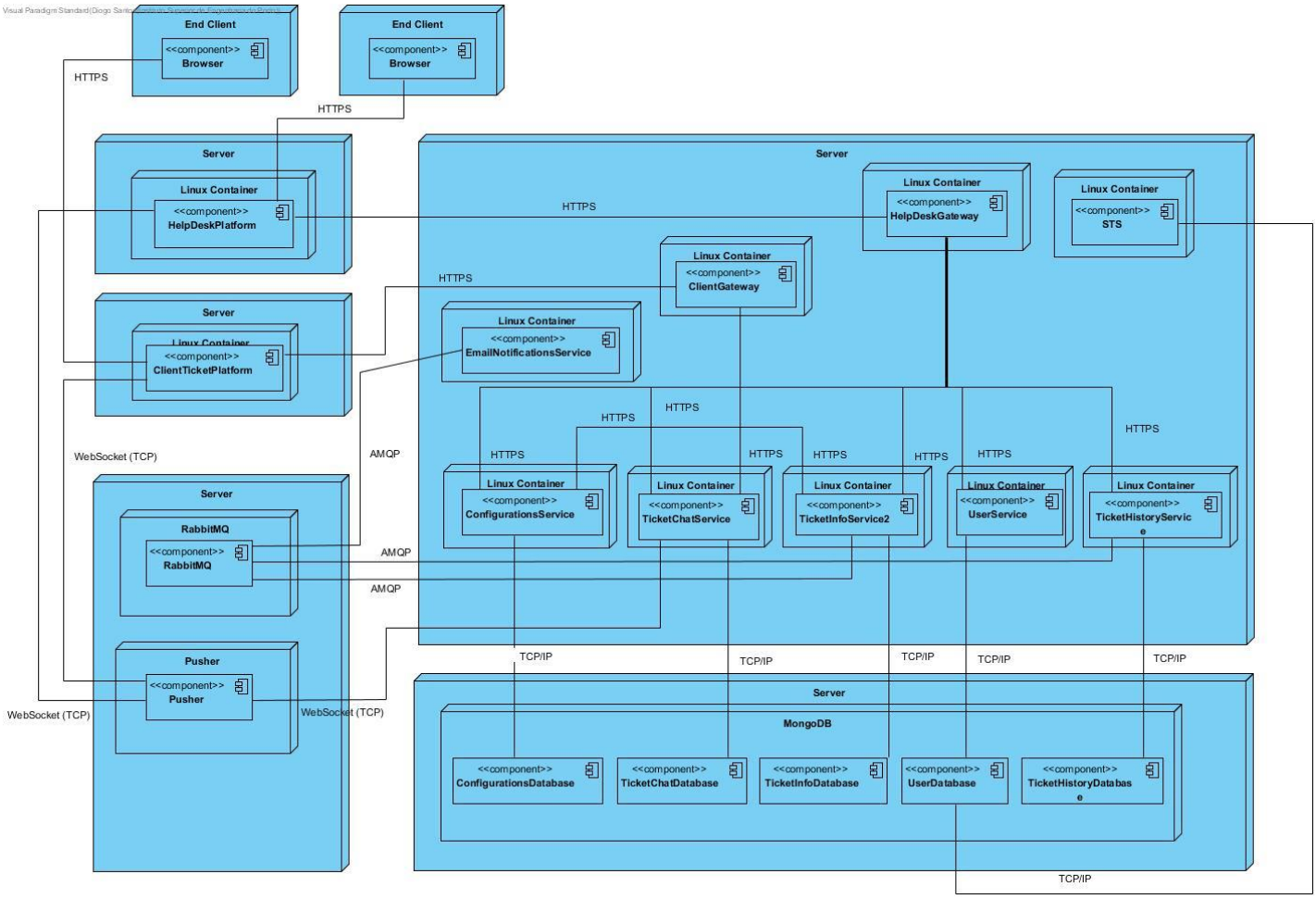


Figura 21 - Diagrama de Implantação

Como é possível observar, existem cinco servidores e dois. O browser é onde ambas as plataformas vão ser utilizadas, embora estas sejam armazenadas em servidores distintos. Em seguida, existe um servidor para os serviços e *gateways*, outro para as bases de dados e um último para os clientes de "Pusher" e "RabbitMQ". Existem comunicações que foram omitidas do diagrama para tentar simplificar este, como por exemplo:

- Comunicação entre as plataformas e o "STS";
- Comunicação entre o "STS" e os serviços;
- Comunicações HTTPS entre os serviços.

No servidor dos clientes são representadas as instâncias de "Pusher" e "RabbitMQ", que devem estar instaladas e a correr no servidor com o próprio componente.

Assim como no servidor designado para as bases de dados, deve existir uma instância de "MongoDB" a correr com as bases de dados.

5 Implementação

Neste capítulo é descrita a implementação do sistema e são descritos todos os seus componentes e como estes estão conectados uns com os outros. Também serão descritas as dificuldades encontradas e como foram ultrapassadas.

5.1 Autenticação e Autorização

Tal como foi referido na secção 4.3.2, foi adotado o "Identity Server" para a Autenticação e Autorização do Sistema. Sendo assim, a primeira ação é criar o STS ("Security Token Service"). O STS vai ser o serviço responsável pela geração e validação dos *tokens*.

O STS é uma combinação de *middlewares* e serviços, sendo que toda configuração é feita na classe "Startup.cs". O serviço pode ser criado através do template de uma *Api* em ".NetCore", presente no Visual Studio (IDE utilizado para desenvolver os serviços). Após o Serviço ser gerado, é necessário alterar a classe "Startup.cs" com as configurações necessárias aos requisitos do sistema, indicando a utilização do "Identity Server" e onde ir buscar os dados como "scopes", clientes e utilizadores (utilizadores que podem potencialmente fazer login).

Na parte dos utilizadores encontrou-se o primeiro obstáculo: o próprio "Identity Server" foi desenhado para a base de dados de utilizadores ser lida através de "Entity Framework" e a própria documentação está orientada a isso. No entanto, a base de dados adotada para o projeto foi "MongoDB" e todos os serviços utilizam o "MongoDBClient" para a gestão de dados. Portanto, utilizar "Entity Framework" não seria o ideal.

Para ser possível obter os utilizadores através de um serviço (passando pela camada de lógica antes da camada de dados como foi definido), foi necessário implementar uma extensão à interface "IProfileService". Nesta implementação é possível obter os utilizadores de um serviço e aplicar as *claims* necessárias. É importante referir que, para obter os utilizadores, é feita uma leitura à base de dados "UserDatabase", gerida pelo serviço "UserService".

As *claims* são propriedades que os serviços que recebem o token vão ser capazes de obter e utilizar, tendo em vista os requisitos do sistema. As claims definidas para os utilizadores foram:

- Id;
- Nome;
- E-mail;
- Cargo (Administrador ou Colaborador).

No extrato de código da Figura 22 é possível observar a extensão chamada na classe "Startup.cs", onde foram inseridas estas configurações - incluindo a implementação criada para obter os utilizadores ("Helpers.ProfileService"):

```
        public static IServiceCollection AddAuthComponents(this IServiceCollection
services)
        {
            services.AddIdentityServer()
                .AddDeveloperSigningCredential()
                .AddInMemoryIdentityResources(Config.GetIdentityResources())
                .AddInMemoryApiScopes(Config.GetApiScopes())
                .AddInMemoryClients(Config.GetClients())
                .AddProfileService<Helpers.ProfileService>();

            services.AddTransient<IResourceOwnerPasswordValidator,
Helpers.ResourceOwnerPasswordValidator>();
            services.AddTransient<IProfileService, Helpers.ProfileService>();

            return services;
        }
```

Figura 22 - Configurações STS

Dependendo das funcionalidades do sistema e a sua distribuição, podem ser gerados vários *scopes* - é importante relembrar que os *scopes* servem para a autorização (apesar de também fazerem parte da autenticação). Estes são usados para proteger recursos para clientes que lhes tenham acesso. Na Tabela 13 são descritos todos os clientes criados no STS e os parâmetros utilizados em cada um.

Tabela 13 - Clientes STS

Cliente	Fluxo de Autenticação	Scopes
ticket-info-service	ClientCredentials	- configuration - user - department
ticket_application	ResourceOwnerPassword (Requer login de utilizadores)	- configuration - user - department - ticket - ticket.list - ticket.notes - ticket.history - ticket.chat
client_ticket_application	ClientCredentials	- department - configuration - ticket.client - ticket.create - ticket.chat

Em cada cliente devem apenas ser configurados os *scopes* a que esse cliente deve ter acesso. Algo que se pode observar como exemplo, é que o "ticket_application" que representa a plataforma de *help desk*, não possui o "scope" "ticket.create", pois os tickets apenas podem ser criados do lado do cliente. Portanto, apenas a plataforma utilizada por este, tem acesso a esse "scope".

No final destas configurações o STS está pronto a correr. Para testar, é possível fazer um simples pedido para obter um token. A Figura 23 representa esse pedido, efetuado através da plataforma "Postman" para o cliente "ticket-info-service".

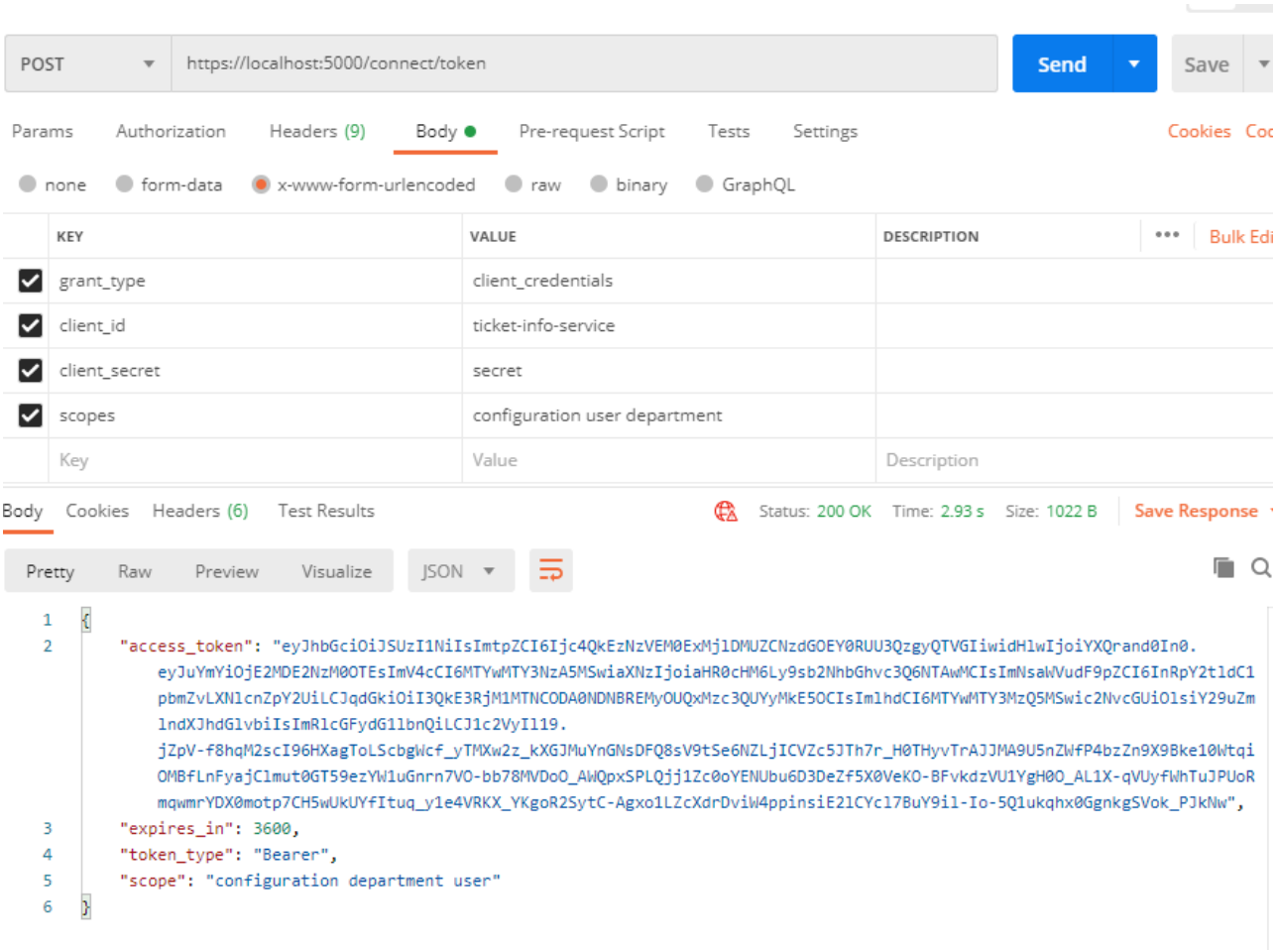


Figura 23- Pedido de token ao STS

Através de um pedido semelhante a este (no fluxo de password é preciso acrescentar o *username* e *password)* é possível os diferentes clientes se autenticarem.

Depois de configurado o STS, está na altura de proteger os endpoints e aqui foi encontrado outro problema. O *Identity Server* tem filtros nativos que implementam estas proteções para *scopes* ou *claims*, no entanto, o que é requisitado, é uma mistura entre ambos, ou seja:

- Caso não haja token -> Retornar 401;
- Caso haja token, verificar se é cliente;
- Caso seja cliente -> validar se o token possui o “scope” necessário, caso não possua-> Retornar 403;
- Caso seja utilizador -> validar se o token possui a *claim* necessária (cargo) -> caso não possua, ou o valor não for o necessário -> Retornar 403.

Como é necessária uma lógica própria para estas validações, foi necessária a implementação de um filtro que efetuasse esta validação. Para isto, foi necessário criar o filtro e um atributo que implementasse esse filtro. Na Figura 24 é possível observar a utilização deste filtro nas linhas 49 e 56 para proteger 2 *endpoints*.

```
[HttpGet("{id}")]
[ScopeAndRoleAuthorization(Scopes.DepartmentScope)]
public ActionResult<Department> Get(string id)
{
    return this._service.Get(id);
}

[HttpPost]
[ScopeAndRoleAuthorization(Scopes.DepartmentScope, ERole.Admin)]
public ActionResult<Department> Create(Department department)
{
    return this._service.Create(department);
}
```

Figura 24- Filtro para Autorização

Como se pode observar, este filtro é utilizado para o *scope departments*, o que significa que qualquer cliente com este "scope" pode aceder a ambos os recursos. No entanto, caso não seja um cliente a tentar aceder, mas um utilizador, existe um exemplo da validação por cargo. No *endpoint* "Create", é possível observar na linha 56, que está a ser enviado o parâmetro do cargo para o filtro. Desta forma, caso seja um utilizador, ele vai validar se o cargo desse utilizador é igual ao enviado. O que significa que, caso um colaborador tente criar um departamento, este vai ter a resposta 403 - que indica que não está autorizado a fazê-lo.

Existem outros casos específicos, onde é preciso utilizar mais parâmetros do token (por exemplo, ao obter os tickets, é necessário, que caso seja um colaborador, retornar apenas os tickets desse colaborador, o que é possível obtendo a *claim* de "user_id"). Estes casos específicos são implementados individualmente.

Ao utilizar este filtro em todos os serviços, é garantida uma total proteção a todos os recursos. Ao mesmo tempo, também é facilitada a escalabilidade e integrabilidade do sistema. Caso seja preciso criar um cliente novo interno ou externo, pode ser realizado através de uma simples alteração ao STS. Existem várias formas de agilizar este processo, alguns exemplos são: Utilizar uma base de dados para guardar os clientes com um serviço próprio para a gerir ou ao invés de utilizar uma classe interna, utilizar um ficheiro de configurações para guardar os clientes. Este ficheiro poderia ser facilmente alterado sem ser necessário uma alteração de versão e nova publicação do STS. No entanto, os requisitos do sistema não validavam estas implementações, pelo que foi adotada a abordagem indicada na própria documentação do *Identity Server*, de utilizar uma classe "Config.cs".

Quanto á plataforma, na Figura 25 é possível observar a página de login para os administradores e colaboradores:

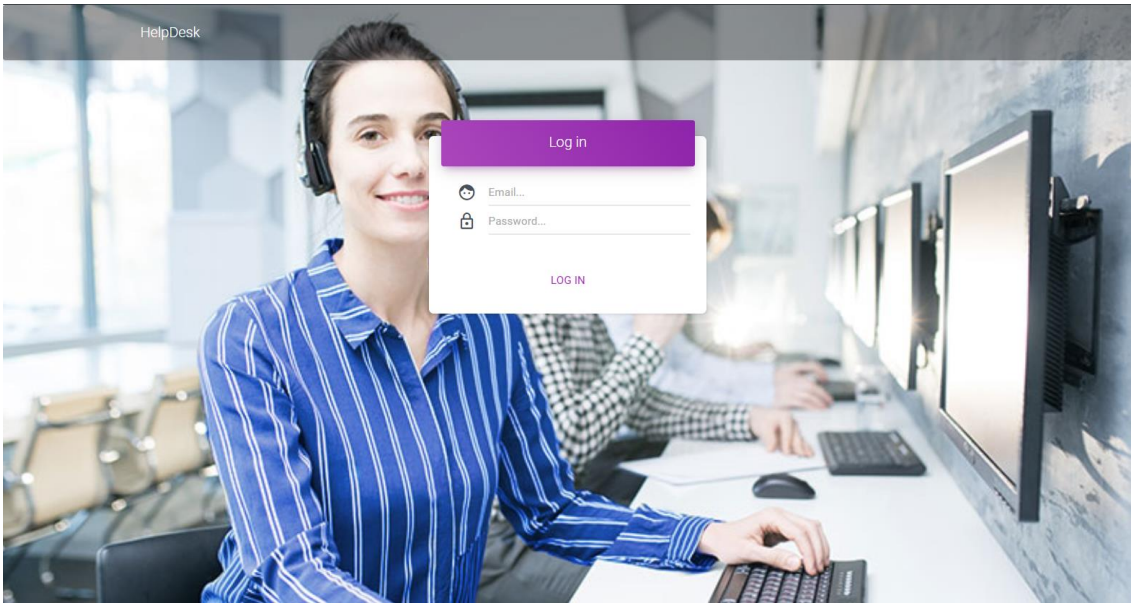


Figura 25 - Página de Autenticação

5.2 Estrutura dos serviços

Os componentes mais importantes para este sistema, são os serviços. Tendo isso em conta, é importante explicar como estes foram desenvolvidos e a estrutura que possuem. Para servir de exemplo, irá ser utilizado o serviço mais complexo do sistema, o "TicketInfoService".

Na Figura 26 é possível observar a estrutura de pastas e projetos deste serviço:

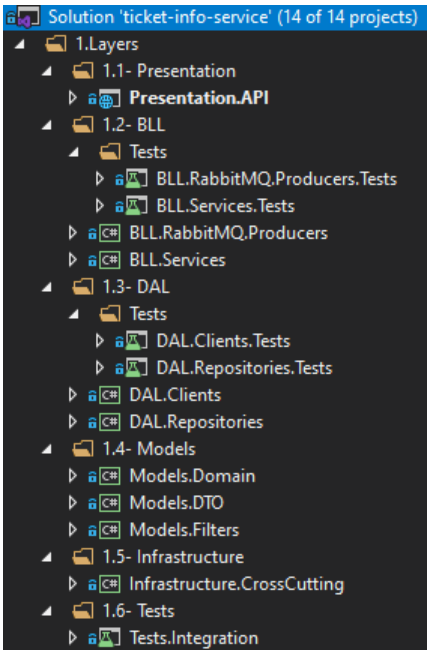


Figura 26 - Estrutura "TicketInfoService"

Como é possível observar, a solução é dividida em pastas para cada camada apresentada na secção 4.2.1.3, sendo que existe uma pasta adicional - “Tests” - para os testes de integração. Também são criados projetos diferentes dentro de cada camada, um para cada responsabilidade, deixando os projetos o mais desacoplados possível. Um exemplo disto é a camada “Models”, que possui um projeto separado para os modelos de domínio, para os modelos que possuem transformações específicas para retornar nos *endpoints* (“DTOs”, *Data Transfer Objects*) e modelos utilizados para filtragens de pesquisas. Com esta estrutura, é possível perceber facilmente a responsabilidade de cada projeto.

5.2.1.1 Injeção de Dependências

Um padrão importante, adotado no desenvolvimento dos serviços, foi o padrão de Injeção de Dependências (*Dependency Injection* ou DI). Através deste padrão é possível criar as dependências necessárias para cada classe, garantindo que cada classe as recebe sem ser preciso a sua criação de cada vez que é necessário. Para demonstrar este padrão, existe o exemplo das classes necessárias à produção de eventos. O “TicketService.cs”, serviço responsável pela lógica dos tickets necessita, entre outras dependências, dos produtores de eventos de “RabbitMQ”, para chamar os eventos necessários. Usando DI, estas dependências podem ser injetadas no serviço, sem ser necessária a sua criação de cada vez que o serviço é chamado, na Figura 27 é possível ver como estas dependências são obtidas e usadas pelo serviço:

```
private ITicketRepository _repo;
private ITicketReassignService _reassignService;
private ITicketStateChangedEventProducer _stateChangedProducer;
private ITicketCreatedEventProducer _ticketCreatedEventProducer;
private ITicketFieldsUpdatedEventProducer _ticketFieldsUpdatedProducer;

public TicketService(ITicketRepository _repo
      , ITicketStateChangedEventProducer stateChangedProducer
      , ITicketReassignService reassignService
      , ITicketCreatedEventProducer ticketCreatedEventProducer
      , ITicketFieldsUpdatedEventProducer ticketFieldsUpdatedProducer)
{
    this._repo = _repo;
    this._reassignService = reassignService;
    this._stateChangedProducer = stateChangedProducer;
    this._ticketCreatedEventProducer = ticketCreatedEventProducer;
    this._ticketFieldsUpdatedProducer = ticketFieldsUpdatedProducer;
}
```

Figura 27 - DI no "TicketService"

Como é possível observar, estas dependências são injetadas no construtor e utilizadas pelo serviço. Por sua vez, o mesmo é realizado para injetar o serviço no “Controller” ou “Controllers” que precisam do mesmo. Para ser possível injetar estes produtores é necessário adicioná-los ´instância de “IServiceCollection” definida na classe “Startup.cs”. Na Figura 28 é possível ver como é feita essa adição:

```
        public static IServiceCollection AddRabbitMQProducers(this IServiceCollection
services)
        {
            services.AddScoped<ITicketStateChangedEventProducer>(p => new
TicketStateChangedEventProducer(p.GetService<IOptions<RabbitMQSettings>>().Value));
            services.AddScoped<ITicketCreatedEventProducer>(p => new
TicketCreatedEventProducer(p.GetService<IOptions<RabbitMQSettings>>().Value));
            services.AddScoped<ITicketReassignedEventProducer>(p => new
TicketReassignedEventProducer(p.GetService<IOptions<RabbitMQSettings>>().Value));
            services.AddScoped<ITicketFieldsUpdatedEventProducer>(p => new
TicketFieldsUpdatedEventProducer(p.GetService<IOptions<RabbitMQSettings>>().Value));

            return services;
        }
```

Figura 28- Declaração dos produtores de RabbitMQ

Desta forma, sempre que for injetada uma instância da interface "ITicketStateChangedProducer" irá ser utilizada a instância definida de "TicketStateChangedEventProducer". Existem 3 métodos que podem ser chamados para declarar as dependências. O presente na Figura 28 é o "AddScoped":

- "AddScoped": Sempre que for necessária uma nova instância, esta vai ser novamente criada. Ou seja, caso duas classes utilizem a mesma dependência, vai ser criada uma instância para cada uma.
- "AddTransient": Funciona por pedido a "endpoint". Ou seja, caso no mesmo pedido 2 classes utilizem a mesma dependência, essa vai ser a mesma instância. No entanto, num novo pedido, são criadas instâncias novas.
- "AddSingleton": Utiliza o padrão de *singleton*. Ou seja, vai ser criada apenas uma instância da classe e essa mesma instância vai ser sempre injetada em todas as classes que a necessitem. É utilizado, maioritariamente, para as configurações do projeto.

No entanto, é possível observar que estas dependências são declaradas numa extensão - foi adotada esta alternativa para simplificar a classe "Startup.cs". É possível ver a chamada das diferentes extensões através do padrão *builder*, na Figura 29.

```
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddSettings(Configuration) //Adds system configuration classes
                .AddRepositories() //Adds repositories and MongoDB connection
                .AddServices() //Adds services
                .AddClients() //Adds HttpClients
                .AddRabbitMQProducers(); //Adds RabbitMQ Producers
```

Figura 29- Declaração de Dependências na classe "Startup.cs"

5.2.1.2 Configurações de sistema

Existem diversos parâmetros necessários para algumas funcionalidades do Sistema, como por exemplo a ligação à base de dados e ligação a clientes como "RabbitMQ". No entanto, estas configurações podem depender de diversos fatores como ambiente (a ligação á base de dados de desenvolvimento é diferente da base de dados de produção). Devido a isto, estas

configurações não podem estar definidas dentro do código. A alternativa adotada e recomendada, é a sua definição num ficheiro *Json* de configurações - normalmente denominado "appsettings". Neste ficheiro é possível definir, no formato *Json*, as configurações necessárias. Posteriormente, o sistema consegue ler esse ficheiro e é utilizado para mapear para classes do sistema e injetá-las onde for necessário (como *singletons*).

Na Figura 30 é possível observar o ficheiro de configurações para o "TicketInfoService":

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "MongoDBConnection": {
    "Database": "ticket-info-db",
    "ConnectionString": "mongodb://localhost:27017"
  },
  "RabbitMQSettings": {
    "Host": "localhost",
    "Port": 5672,
    "Username": "ticketapp",
    "Password": "ticketapp"
  },
  "UriSettings": {
    "ConfigurationServiceUri": "https://localhost:5004",
    "UserServiceUri": "https://localhost:5002"
  },
  "AuthenticationSettings": {
    "Authority": "https://localhost:5000",
    "Client_Id": "ticket-info-service",
    "Client_Secret": "secret",
    "Scopes": "configuration user department"
  }
}
```

Figura 30 - "appsettings" do "TicketInfoService"

Neste ficheiro, é possível observar algumas configurações pré-definidas para serem utilizadas pela framework adotada (no ambiente de desenvolvimento) e outras adicionadas pela equipa de desenvolvimento, devido ao facto de serem necessárias para o sistema:

- "MongoDBConnection": definições para conexão á base de dados;
- "RabbitMQSettings": definições para conexão ao RabbitMQ;
- "UriSettings": definição de ligações a outros serviços necessários;
- "AuthenticationSettings": Tal como foi referido, o "TicketInfoService" necessita de realizar alguns pedidos HTTP a outros serviços. Como tal, é preciso autenticar-se no STS. Estas configurações são as utilizadas para essa autenticação.

5.3 Messaging

Uma parte importante para o funcionamento correto do sistema, são os eventos. Estes são usados como forma de comunicação assíncrona entre os micro serviços. Como tal, é importante que estejam bem configurados e funcionem corretamente.

O "RabbitMQ", cliente utilizado como *message broker* do sistema, funciona através de *exchanges* e queles. Definindo uma *queue*, é possível enviar e ler mensagens dessa mesma *quele*. Sendo assim, para cada *queue* existe um produtor e um consumidor. Nesta secção, será descrito como ambos são configurados e qual o funcionamento de ambos. Para isso, será utilizado como exemplo, o evento de alteração de estado de um ticket - um dos mais importantes e mais utilizado do sistema - que tem como produtor o "TicketInfoService" e como consumidor o "TicketHistoryService".

O Produtor é o serviço responsável por enviar a mensagem. Na Figura 31 é possível observar o método que envia a mensagem:

```
public async Task Produce(TicketStateChangedEventBody message)
{
    try
    {
        using (var connection = factory.CreateConnection())
        using (var channel = connection.CreateModel())
        {
            channel.QueueDeclare(queue: QUEUENAME,
                                 durable: false,
                                 exclusive: false,
                                 autoDelete: false,
                                 arguments: null);

            byte[] body = Encoding.Default.GetBytes(JsonConvert.SerializeObject(message));

            channel.BasicPublish(exchange: "",
                                 routingKey: QUEUENAME,
                                 basicProperties: null,
                                 body: body);

            Console.WriteLine($"TicketStateChanged Message published with success!");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Something Went Wrong Publishing TicketStateChangedEvent! {ex.Message}");
    }
}
```

Figura 31- Produzir mensagem

Como é possível observar, primeiramente é estabelecida a conexão e um canal para comunicar com o "RabbitMQ"; depois é declarada a *queue* no canal, caso esta não exista, é criada neste passo. Após ter a *queue* declarada, é feito o *publish* com a mensagem no binário. Depois da mensagem ser enviada, o canal e conexão são descartados.

A consumidor funciona quase da mesma forma, tendo as seguintes diferenças:

- Em vez de o *publish*, é realizado o *consume* e enviado como parâmetro a função a chamar, quando uma mensagem for recebida;
- O Canal e a conexão nunca são descartados; o consumidor usa-os enquanto estiver a correr. Caso estes sejam descartados, não é possível continuar a consumir mensagens.

Existem 4 *queues* no sistema. Na Tabela 14 são descritas cada uma delas, indicando os produtores e consumidores e função do evento:

Tabela 14 - Eventos do Sistema

Queue	Produtor	Consumidor	Função
TicketStateChangedEvent	TicketInfoService	TicketHistoryService	Guardar ocorrência de alteração de estado no histórico do ticket.
TicketFieldsUpdatedEvent	TicketInfoService	TicketHistoryService	Guardar ocorrência e campos alterados no histórico do ticket.
TicketReassignedEvent	TicketInfoService	TicketHistoryService	Guardar ocorrência de alteração de colaborador no histórico do ticket.
TicketCreatedEvent	TicketInfoService	TicketHistoryService	Guardar ocorrência de criação de ticket no histórico deste.

5.4 Casos de Uso

Nesta secção é descrita a implementação dos casos de uso mais relevantes do sistema, analisados e desenhados na secção 4.3.3.

5.4.1 UC07- Criar um ticket

Este caso de uso envolve principalmente o “ClientTicketPlatform” e o “TicketInfoService”.

Quando o cliente entra na plataforma, existem duas ações a ser feitas: guardar as informações do cliente e autenticar a aplicação. Podemos utilizar a forma como o Angular se comporta, para realizar ambas em componentes diferentes, separando assim as responsabilidades.

O Angular funciona através de componentes, pelo que é possível colocar componentes dentro de outros. O próprio roteamento funciona como um componente - é necessário colocar a sua *tag* em html para ser usado - por norma esta *tag* é colocada dentro do componente "app.component" – por isso, este componente vai existir sempre, independentemente da página onde o utilizador da plataforma se encontrar. Devido a isto é o componente certo para tratar da autenticação.

Para guardar os dados do cliente, é possível criar um componente que possa ser acedido através da rota "/entry/{userId}/{userName}/{userEmail}". Este componente pode, de imediato, guardar estes dados e redirecionar o cliente para a *dashboard* da plataforma.

Este fluxo pode ser observado no diagrama de sequência da Figura 32:

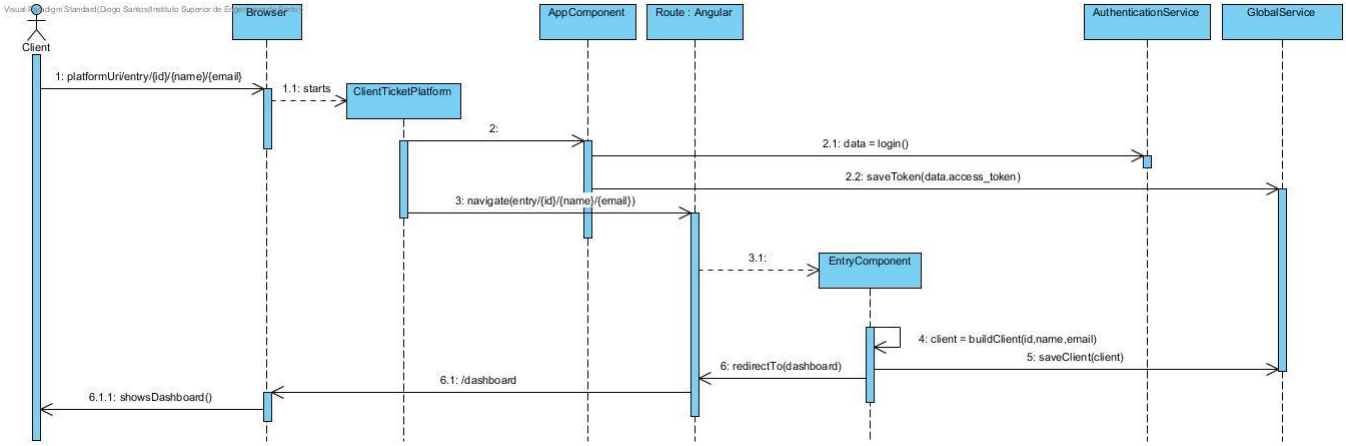


Figura 32 - Fluxo acesso a ClientTicketPlatform

Desta forma, qualquer plataforma pode integrar este sistema e enviar o utilizador para a plataforma de tickets, através de um *url*.

Na *dashboard*, é apresentada ao utilizador a escolha de criar um ticket.

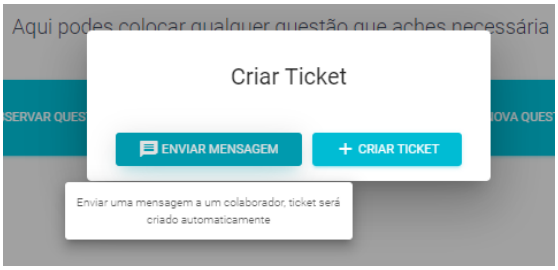


Figura 33 - Caixa de Diálogo Criar Ticket

Quando o utilizador seleciona a opção de criar ticket, é apresentada a caixa de diálogo da Figura 33, que pergunta ao utilizador se este pretende criar um ticket ou apenas enviar mensagem. Na secção de enviar mensagem existe uma *tooltip* que descreve o que essa opção significa.

Após selecionar uma opção, a aplicação é redirecionada para a página correspondente, onde o utilizador pode preencher os campos necessários e criar o ticket ou enviar a mensagem. O resto da implementação segue o diagrama de sequência da Figura 17, até chegar à parte de atribuir o ticket.

Para esta funcionalidade foi criada uma classe separada no "TicketInfoService". Esta classe tem como responsabilidade todas as ações relativas à reatribuição ou atribuição de colaboradores ao ticket. Neste serviço é obtido o departamento configurado como ponto de entrada para os tickets e é chamado o método representado na Figura 34, onde se pode observar a lógica de atribuição do ticket a um colaborador.

```
public async Task AssignTicket(Ticket model, Department department, User collaborator)
{
    try
    {
        if (collaborator == null)
            collaborator = await this.GetCollaboratorFromDepartment(department).ConfigureAwait(false)
                ?? throw new Exception("No Collaborator found");
        var isAssigned = model.State == ETicketState.Assigned;
        model = model.AssignCollaborator(collaborator);
        this._ticketRepository.Update(model.Id, model);
        _ = this._ticketReassignedEventProducer.Produce(TicketReassignedEventBody.BuildMessage(model));
        if (!isAssigned)
            _ = this._stateChangedProducer.Produce(TicketStateChangedEventBody.BuildMessage(model));
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Ocorreu um erro ao atribuir o ticket: {ex.Message}");
        throw;
    }
}
```

Figura 34 - Atribuição de ticket a um colaborador

Como é possível analisar, este método é assíncrono, pelo que o objetivo da sua utilização neste fluxo é ser chamado assincronamente, por isso, quando este método estiver a correr, o utilizador já recebeu a confirmação do ticket criado e já foi redirecionado para a lista de tickets. Também se pode reparar que ambas as tarefas de enviar mensagem também são chamadas assincronamente, para manter os serviços desacoplados. É importante que não se dependa do sucesso ou insucesso do envio da mensagem para continuar o fluxo.

Quando o colaborador é alterado no ticket, este altera também o departamento para o do colaborador e o estado para Atribuído.

Para obter o colaborador com maior disponibilidade, são percorridos todos os colaboradores e é obtida a pontuação de disponibilidade destes. No final, retorna-se o colaborador mais disponível (menos pontuação). A pontuação de um colaborador pode ser obtida através da soma do inverso da prioridade de cada serviço, como demonstra a seguinte fórmula:

$$p = \sum(\frac{1}{prioridade})$$

Desta forma, caso existissem os seguintes casos:

- António: Serviço com prioridade 1; Serviço com prioridade 3;
- João: Serviço com prioridade 1;
- Maria: Serviço com prioridade 4; Serviço com prioridade 4; Serviço com prioridade 4.

A pontuação de cada seria:

- António: 1.33;
- João: 1;
- Maria:0.75.

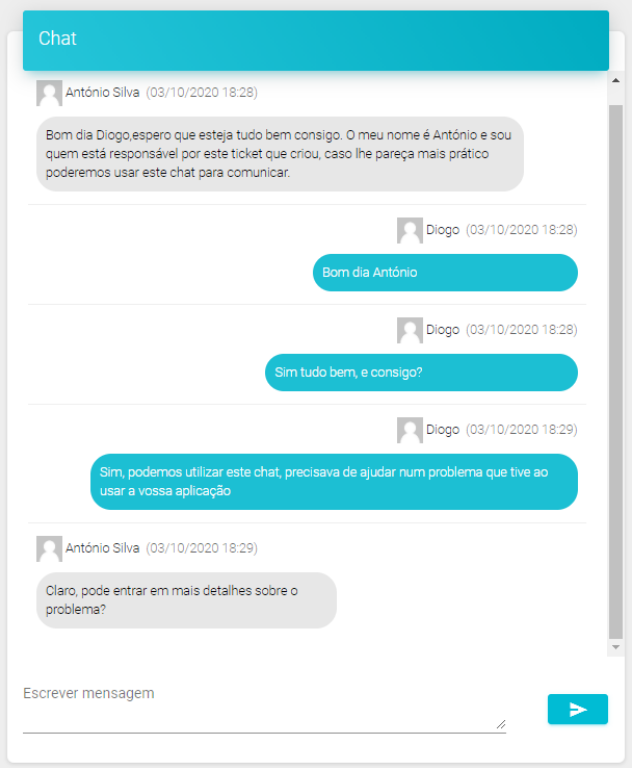
Neste caso, apesar da Maria ser a colaboradora com mais serviços, é a que tem maior disponibilidade, devido à baixa prioridade dos mesmos. Portanto, seria a Maria a ser alocada ao novo serviço.

5.4.2 UC08- Enviar mensagem por chat

Este caso de uso envolve, principalmente, ambas as plataformas, e o "TicketChatService".

Para implementar um chat em tempo real, foi adotado um cliente chamado "Pusher" (https://pusher.com/) especialmente devido ao facto da extensa documentação que este possui - nomeadamente possui documentação para as plataformas que vão interagir com o mesmo - como "Angular" e ".NetCore".

O chat de um ticket está presente dentro da pasta do mesmo, em ambas as plataformas. Basta aceder à página do ticket para ter acesso ao chat do mesmo. O chat pode até mesmo ter vários intervenientes, por exemplo, caso o colaborador do ticket seja alterado, as mensagens antigas continuarão a aparecer com o nome do colaborador antigo.



A Figura 35 representa um chat entre o colaborador António e o cliente Diogo. Neste caso está na perspetiva do Cliente, no entanto na perspetiva do colaborador a interface é idêntica. É possível observar que cada mensagem contém o nome de quem enviou, a data e hora do envio e a mensagem. Também existe um barra de *scroll*, para ser possível ver as mensagens mais antigas - é possível em qualquer ticket ver o chat desde o seu início.

Em baixo, existe uma caixa para escrever a mensagem e um botão de envio.

Figura 35 - Chat

Uma das vantagens de utilizar "Angular" é poder adotar várias bibliotecas que existem online, desenvolvidas por empresas ou mesmo pessoas individuais. Ao início, a ideia para desenvolver o chat passava por adotar uma destas bibliotecas (ou componentes) para a interface do chat. No entanto, isto provou ser um obstáculo. A equipa de desenvolvimento procurou bastante por uma interface que cumprisse todos os requisitos. Ao início foi encontrada uma, no entanto, ao implementar, foi descoberto que existia um problema nesta interface (já reportado e em fase de correção) que impedia o seu uso na versão mais recente de Angular (que por coincidência foi a versão utilizada para o projeto). Devido a isto, foi tomada a decisão de implementar a interface de raiz, ao que resultou a interface representada na Figura 35.

Na Figura 36 é possível observar a inclusão do chat na página do ticket, na plataforma de *help desk*, vista por um colaborador:

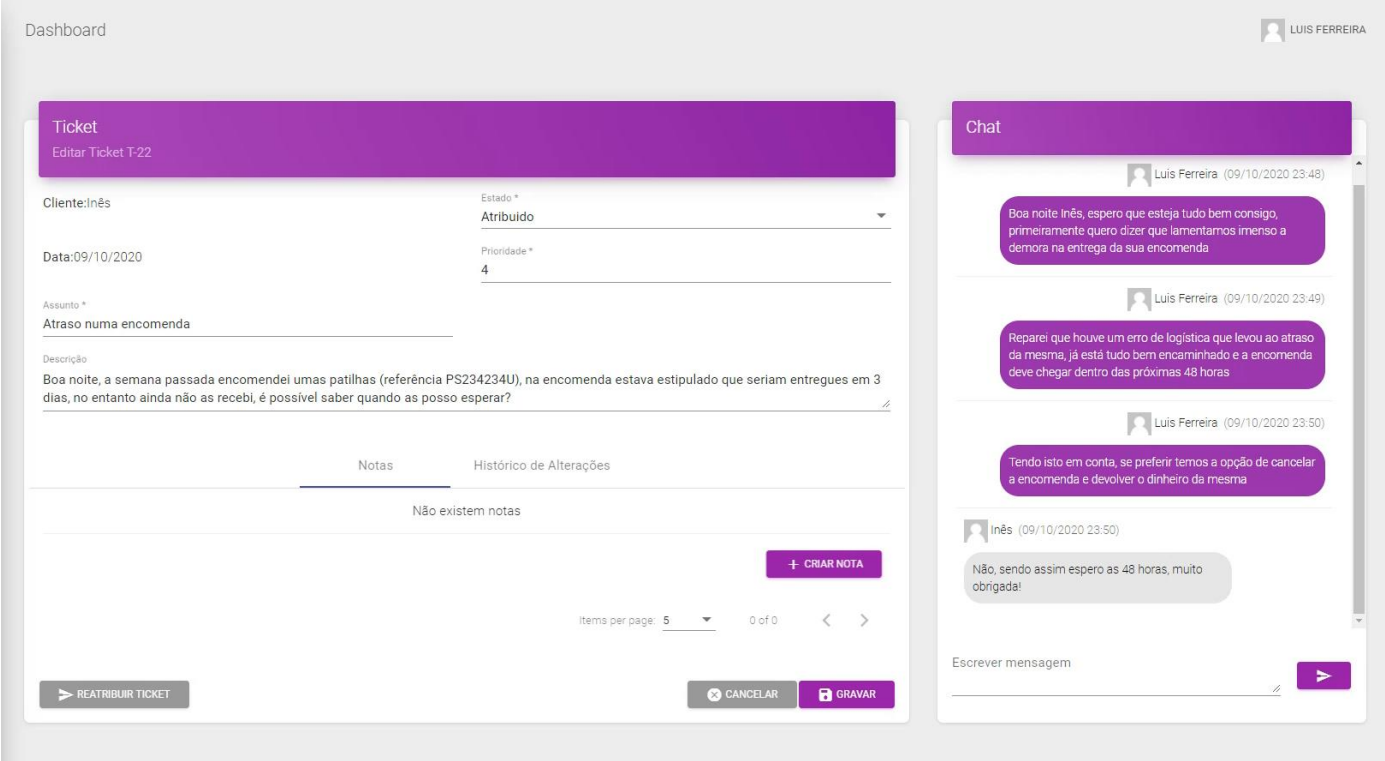


Figura 36- Ticket da vista de um colaborador

Quando o cliente (ou o colaborador) acede à página do ticket, é feito um pedido ao "TicketChatService" para obter todas as mensagens existentes na base de dados, que depois são apresentadas no chat. Após este pedido, é necessário iniciar o consumidor ao "Pusher", para ser capaz de consumir as mensagens em tempo real. Na Figura 37 é possível observar o excerto de código que realiza esta função:

```
setupPusher(ticket_id: string) {
  Pusher.logToConsole = true;

  var pusher = new Pusher('d0d0a4a4c182668bef53', {
    cluster: 'eu'
  });

  var channel = pusher.subscribe('ticket-app');
  channel.bind('ticket-' + ticket_id, ($data) => this.messageReceived(JSON.parse($data.message, function (prop, value) {
    if (!prop)
      return value;
    var lower = prop[0].toLowerCase() + prop.substring(1, prop.length);
    if (prop === lower) return value;
    else this[lower] = value;
  })));
}

messageReceived(message: ChatMessage) {
  if (!this.messagesList.find(m => m.id == message.id)) {
    this.messagesList.push(message);
  }
}
```

Figura 37- Consumidor do "Pusher"

Instalando o módulo do “Pusher” para “Angular”, é possível ter acesso às classes deste cliente, que nos permitem subscrever a uma fila de mensagens.

Como é possível observar, o “Pusher” é iniciado através de uma chave única (fornecida pelo mesmo aquando da criação de uma conta). Existe apenas um canal aberto para o sistema (“ticket-app”). Nesse canal, é possível obter as mensagens relativas a um ticket, pesquisando por “ticket-{id do serviço}”. Com esta configuração, todas as mensagens que foram publicadas para este canal, com o id deste ticket, vão ser lidas neste consumidor. À mesma função do “bind”, é necessário fornecer a função a ser chamada, quando uma mensagem for recebida. Esta função apenas adiciona a mensagem à lista atual de mensagens, e é mostrada ao utilizador da plataforma (cliente ou colaborador). É importante referir que este código e o da interface gráfica é idêntico em ambas as plataformas (na interface gráfica é apenas alterado o lado onde são apresentadas as mensagens, para mostrar as enviadas sempre do lado direito e as recebidas do lado esquerdo).

O “Pusher” também disponibiliza uma pequena *dashboard*, onde é possível demonstrar o estado do canal - apresentada na Figura 38

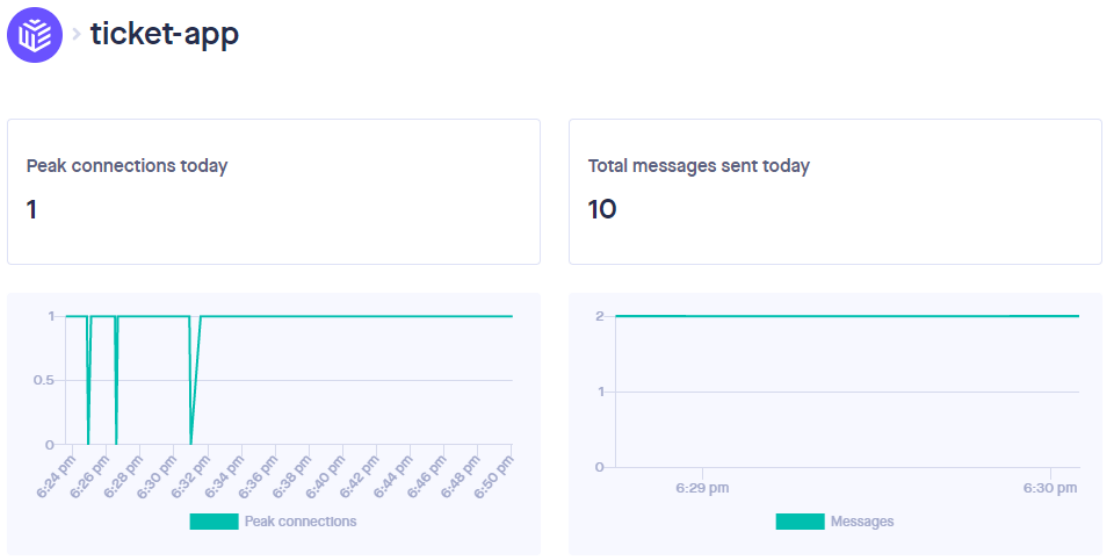


Figura 38- *Dashboard* “Pusher”

Nesta *dashboard* é possível observar o número de consumidores ativos e o total de mensagens enviadas no dia. Olhando, por exemplo, para o gráfico do lado direito, podemos observar que o utilizador saiu da página do ticket nos instantes dos picos apresentados, deitando assim abaixo o consumidor e depois voltando a ativar o mesmo, quando acedeu ao ticket novamente.

Como já foi referido, a responsabilidade de publicar as mensagens no “Pusher” é do “TicketChatService”, tal como demonstra o fluxo da Figura 39:

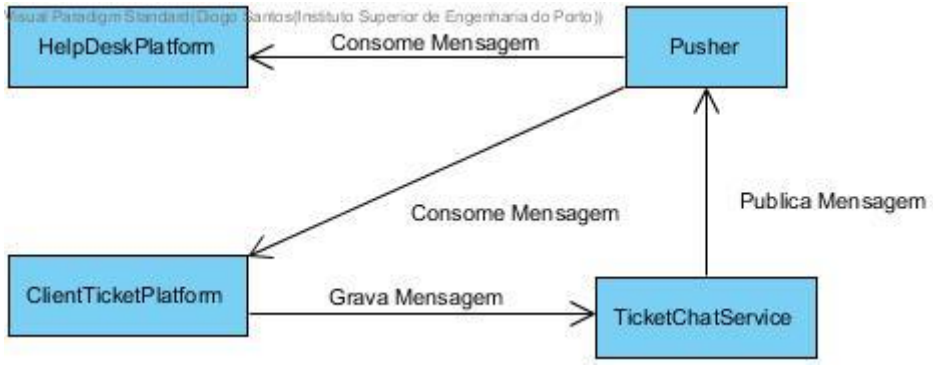


Figura 39- Fluxo Guardar Mensagem

Na Figura 40 é possível observar a classe responsável por produzir a mensagem no “TicketChatService”.

```
public async Task ProduceMessage(ChatMessage message)
{
    var options = new PusherOptions
    {
        Cluster = pusherSettings.Cluster,
        Encrypted = pusherSettings.Encrypted
    };

    var pusher = new Pusher(
      pusherSettings.AppId,
      pusherSettings.AppKey,
      pusherSettings.AppSecret,
      options);

    var result = pusher.TriggerAsync(
      pusherSettings.Channel,
      $"ticket-{message.TicketId}",
      new { message = JsonConvert.SerializeObject(message) });
}
```

Figura 40 - "TicketChatMessageProducer.cs"

Esta classe recebe as configurações do “Pusher” por DI, cria uma instância do “Pusher” com estas configurações e produz uma mensagem com a mesma chave lida pelas plataformas.

5.5 Gateways

Um dos principais objetivos da implementação de Gateways é evitar a necessidade de conhecer a estrutura do sistema para aceder ao mesmo. Com um Gateway, existe apenas um ponto de acesso entre o Cliente e a infraestrutura do sistema. Se não fossem desenvolvidos Gateways para esta solução, as plataformas teriam de conhecer todos os serviços e saber como aceder e o que obter de cada um. Ao implementar um Gateway, garantimos que as plataformas apenas necessitam de conhecer esse ponto de acesso único.

Neste sentido, foram desenvolvidos 2 Gateways - um para cada plataforma.

Os Gateways foram desenvolvidos através da *framework* “Ocelot”. Esta *framework* é especialmente orientada a sistemas com micro serviços em “.NetCore”, pois ela própria é desenvolvida em *.NetCore*. Esta framework lê um ficheiro *Json*, que possui todas as configurações de roteamentos. É neste ficheiro único que é possível configurar os redirecionamentos que o Gateway tem de fazer, logo, é apenas necessário alterar este ficheiro. Na Figura 41 está representado um excerto deste ficheiro para o *HelpDeskGateway*:

```
{
  "DownstreamPathTemplate": "/api/ticket/{ticket_id}/chatMessage/{everything}",
  "DownstreamScheme": "https",
  "DownstreamHostAndPorts": [
    {
      "Host": "localhost",
      "Port": 5010
    }
  ],
  "UpstreamPathTemplate": "/api/ticket/{ticket_id}/chatMessage/{everything}",
  "UpstreamHttpMethod": [ "Get", "Post", "Put", "Delete" ]
},
{
  "DownstreamPathTemplate": "/api/ticket/{everything}",
  "DownstreamScheme": "https",
  "DownstreamHostAndPorts": [
    {
      "Host": "localhost",
      "Port": 5006
    }
  ],
  "UpstreamPathTemplate": "/api/ticket/{everything}",
  "UpstreamHttpMethod": [ "Get", "Post", "Put", "Delete" ]
}
```

Figura 41 - Configurações “Ocelot”

Os roteamentos são configurados através de vários campos:

- *DownstreamPathTemplate*: template do url do serviço para onde este pedido vai ser redirecionado;
- *DownstreamScheme*: Protocolo utilizado pelo serviço;
- *DownstramHostAndPorts*: DNS e porta do serviço;
- *UpstreamPathTemplate*: template do url a redirecionar.
- *UpstreamHttpMethod*: lista de métodos HTTP suportados por este redirecionamento;

Com a configuração existente, é mapeado o endereço: “{gateway_url}/api/ticket/123/chatMessage/Search” para “https:localhost:5010/api/ticket/123/chatMessage/Search”, sendo que o a porta 5010 é a porta onde o *TicketChatService* está configurado para correr.

O redirecionamento para este serviço é configurado antes do redirecionamento para o *TicketInfoService*, para garantir que todos os *urls* iniciados com “/api/ticket” e que não são iguais aos definidos acima, são redirecionados para o *TicketInfoService* (porta 5006).

5.6 Testes

Um dos requisitos não funcionais do sistema era a confiabilidade. Uma das formas de melhorar este requisito é a elaboração de bibliotecas de testes para as diversas funcionalidades, desta

forma, é possível garantir que se alguma funcionalidade não estiver corretamente implementada ou tiver problemas, os testes vão acusar essas falhas.

Foram implementados 2 tipos diferentes de testes: unitários e de integração.

5.6.1 Testes Unitários

Os Testes unitários têm como função testar métodos individuais; estes são aplicados na camada de lógica do sistema e permitem avaliar se o funcionamento de um determinado método é correto.

Para exemplificar um teste unitário, irá ser utilizado um teste para o fluxo do método responsável por obter o colaborador mais disponível (exemplificado na secção 5.4.1).

```
        [TestCategory("Unit")]
        public void GetMostFreeCollaborator_Test3_Success()
        {
            //Setup
            var ticketP1 = this.GenerateTicket();
            var ticketP3 = this.GenerateTicket();
            var ticketP4 = this.GenerateTicket();
            ticketP1.Priority = 1;
            ticketP3.Priority = 3;
            ticketP4.Priority = 4;

            var antonio = this.GenerateUser();
            antonio.Name = "António";
            var joao = this.GenerateUser();
            joao.Name = "João";
            var maria = this.GenerateUser();
            maria.Name = "Maria";

            this.repoMock.Setup(r => r.Search(It.Is<TicketFilter>(f=>f.CollaboratorId == antonio.Id)))
                .Returns(new List<Ticket>() { ticketP1, ticketP3 });
            this.repoMock.Setup(r => r.Search(It.Is<TicketFilter>(f => f.CollaboratorId == joao.Id)))
                .Returns(new List<Ticket>() { ticketP1 });
            this.repoMock.Setup(r => r.Search(It.Is<TicketFilter>(f => f.CollaboratorId == maria.Id)))
                .Returns(new List<Ticket>() { ticketP4, ticketP4, ticketP4 });

            //Act
            var result = this.target.GetMostFreeCollaborator(new List<User>() { antonio, joao, maria });

            //Assert
            Assert.AreEqual(maria.Id, result.Id, $"Colaborador mais livre é {result.Name} em vez da
Maria");
        }
```

Figura 42 - Exemplo de Teste unitário

Analisando o teste da Figura 42, é possível observar que existem 3 fases no teste.

A fase de *setup* é utilizada para criar e configurar todas as variáveis necessárias para o teste. Neste exemplo, primeiro são criados 3 tickets, com prioridades diferentes; de seguida, são criados 3 utilizadores com nomes diferentes. Como o método utilizado obtém os serviços através da base de dados, é necessário simular este pedido com uma resposta. O que se pretende aqui é configurar o método *Search* para retornar os tickets que pretendemos,

dependendo do colaborador. Ao António, por exemplo, é atribuído um ticket de prioridade 1 e outro de prioridade 3.

A fase *Act* é utilizada para a ação pretendida pelo teste, que neste caso é chamar o método e obter o colaborador mais livre.

Na fase *Assert* verifica-se se o teste correu como pretendido, neste caso, a resposta correta, seria a Maria (justificado na secção 5.4.1) e aqui é validado se foi esse o colaborador retornado.

Noutros testes que possuem esse requisito, também é verificado que os clientes e produtores foram chamados.

5.6.2 Testes de Integração

Os testes de integração são utilizados para testar as interações entre diversos componentes e, por isso, são usados para testar fluxos do sistema. Neste caso, foram implementados testes para a interação entre o serviço e a base de dados. Nestes testes é necessário garantir que em cada fluxo, os dados persistidos são os pretendidos.

Para exemplificar um teste de Integração, foi adotado um dos fluxos mais complexos do sistema: o de reatribuir um utilizador. Este teste pode ser visto na Figura 43.

```
        [TestMethod]
        [TestCategory("Integration")]
        public async Task AssignTicket_Success()
        {
            Setup
            var ticket1 = this.fixture.GenerateTicket();
            ticket1.State = ETicketState.Created;
            ticket1 = this._ticketService.Create(ticket1);

            var collaborator = this.fixture.GenerateUser();
            this.userClientMock.Setup(x =>
x.GetCollaboratorsFromDepartment(It.IsAny<Department>())).ReturnsAsync(new List<User> { collaborator });

            //Act
            await this._service.AssignTicket(ticket1).ConfigureAwait(false);

            //Assert
            var updatedTicket = this._ticketService.Get(ticket1.Id);
            Assert.AreEqual(collaborator.Id, updatedTicket.CollaboratorId, "Colaborador não foi
atribuido");
            Assert.AreEqual(collaborator.Department_Id, updatedTicket.DepartmentId, "Departamento não foi
atribuido");
            Assert.AreEqual(ETicketState.Assigned, updatedTicket.State, "Estado não foi alterado para
atribuido");
        }
```

Figura 43 - Exemplo de teste de integração

O objetivo do teste da Figura 43 é garantir que o ticket ficou gravado na base de dados, como era pretendido. Outras validações, como eventos a serem gerados, são testadas através de *Mocks*, pelos testes unitários. Daí não serem vistas neste teste.

Como é possível observar na Figura 43, o *mock* de cliente de utilizadores previamente injetado no serviço, é configurado para retornar apenas o colaborador gerado. Deste modo, garante-se que vai ser este o colaborador a ser atribuído ao ticket. Posteriormente, na fase de *Assert*, valida-se se o colaborador e o departamento foram alterados no ticket e se o estado deste foi alterado para "Atribuído".

5.6.3 Relatório de testes

Tendo em conta os testes apresentados, na Tabela 15 é representado o número de testes para cada serviço:

Tabela 15 - Relatório de Testes

Serviço	Unitários	Integração	Total	Cobertura de Código
TicketInfoService	13	16	29	87.91%
UserService	8	8	16	100%
ConfigurationService	13	13	26	100%
TicketHistoryService	4	4	8	100%
TicketChatService	3	3	6	100%
Total	41	44	85	97.58%

Como é possível observar, devido aos testes de integração serem aplicados na camada de lógica, onde também são realizados os testes unitários, os números de testes são semelhantes. Isto também permite concluir, que cada serviço é simples e conciso.

A cobertura de código, representa a percentagem deste que está a ser coberto pelos testes. Foi definido na secção 4.1.3.6 que o limite para este parâmetro seria de 85% para cada serviço. Todos os serviços cumprem 100% de cobertura, à exceção do "TicketInfoService", que tem 87.91%. Este serviço é o que possui maior lógica, sendo assim, e apesar de todos os métodos estarem a ser testados, podem existir fluxos que não estejam, o que justifica este valor. Apesar deste valor estar dentro das métricas definidas, está perto do limite. Como tal, será melhorado no futuro.

5.7 Pipelines (CI/CD)

Para desenvolver o processo de pipelines foi utilizado o "Azure Devops", plataforma desenvolvida pela Microsoft. É possível utilizar esta plataforma para controlo de versões, para os serviços desenvolvidos nesta solução, o "Github" foi a plataforma utilizada para este controlo, no entanto é possível a integração entre os dois, de forma a que qualquer commit feito para o "GitHub" inicie a pipeline no "Azure Devops".

Para desenvolver estas pipelines foi adotado o serviço "TicketInfoService", devido a ser o mais complexo da solução. Começou-se por integrar o repositório deste serviço com o "Azure Devops", depois criou-se uma pipeline de CI base para este serviço, que corre em todos os commits.

Após a definição de CI foi criada uma *release* (CD), esta é a etapa responsável por fazer *deploy* da plataforma para o "Azure App Services", onde esta vai ser alojada, nesta configuração foi definido um *trigger* automático para iniciar esta *release* após CI correr com sucesso, mas apenas para o *branch* de "master". Após o commit seguinte tudo correu com sucesso e a plataforma foi publicada, na Figura 44 abaixo é possível observar o seu estado no "Azure App Services".

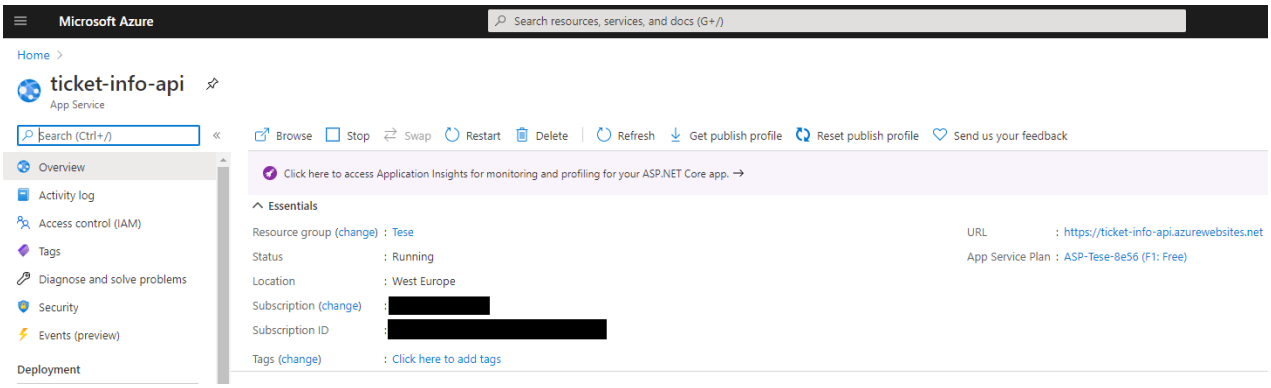


Figura 44- Estado do "TicketInfoService"

No entanto esta pipeline não vai de acordo ao que foi planeado, na Figura 13, nesta á possível observar que o serviço devia ser publicado dentro do

A pipeline de CI criada é uma base apenas e não corre os testes, pelo que são precisas alterações para isto funcionar. Primeiramente é preciso separar os testes de integração dos testes unitários (para que corram em etapas diferentes), para isso é usada a anotação "TestCategory", desta forma separa-se os testes em categorias e é possível correr apenas os testes de uma categoria em cada etapa.

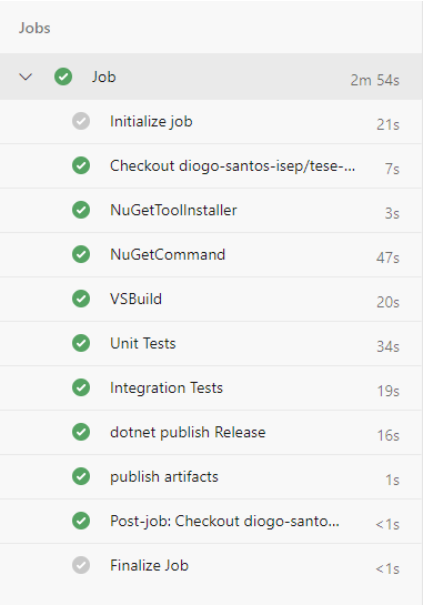


Figura 45- Pipeline CI

Após esta adição os passos que a pipeline corre podem ser observados na Figura 45.

Esta começa por fazer checkout ao *branch* que recebeu o *commit*, posteriormente corre alguns comandos para obter todos os packages e dependências da solução.

De seguida faz build á aplicação e corre os 2 passos de teste adicionados anteriormente, que correm com sucesso.

Os passos seguintes estão relacionados com a preparação da aplicação para ser publicada quando a *release* correr, correndo tudo com sucesso, caso estejamos no *branch* de "master", a pipeline de *release* é iniciada.

6 Avaliação

Neste capítulo pretende-se avaliar o projeto desenvolvido para garantir que cumpre os objetivos pretendidos. Primeiramente é necessário definir as hipóteses da experiência. Depois, é preciso definir como se vai suportar e testar essas hipóteses, referindo os indicadores das mesmas, os testes e experiências a realizar e como estes vão ser avaliados e classificados.

6.1 Hipóteses

Para a avaliação do projeto, devem ser definidas as hipóteses em primeiro lugar. Estas vão comprovar se os objetivos do projeto foram ou não cumpridos.

No âmbito deste projeto existirão 2 hipóteses: a nula (H0) e a alternativa (Ha). Caso seja provada a hipótese nula, implica que os objetivos não foram cumpridos. Caso contrário, comprova-se que os objetivos propostos foram alcançados.

Assim sendo, a hipótese nula é:

H0 = O software desenvolvido não aumenta a eficiência e o controlo do processo de *help desk*, nem aumenta a satisfação do utilizador com a plataforma que está a utilizar.

Esta hipótese pode ser dividida nos seguintes critérios:

- Software não efetua uma gestão eficaz dos tickets;
- Software não facilita o processo de interação entre os utilizadores de uma plataforma e os colaboradores de *help desk*;
- Software não permite um rastreio e visualização dos tickets por parte dos utilizadores.

A hipótese alternativa surge da negação da hipótese H0 e é:

Ha = O software desenvolvido aumenta a eficiência do processo de *help desk* e melhora a interação com os utilizadores aumentando a sua satisfação.

Esta hipótese pode ser dividida nos seguintes critérios:

- Software efetua uma gestão eficaz dos tickets;
- Software facilita o processo de interação entre os utilizadores de uma plataforma e os colaboradores de *help desk*;
- Software permite um rastreio e visualização dos tickets por parte dos utilizadores.

Caso se consiga refutar a hipótese H0, e consequentemente, validar a hipótese Ha, confirma-se que o projeto cumpriu os objetivos pretendidos.

6.2 Indicadores e Fontes de Informação

Para avaliar as hipóteses, é necessário definir primeiro os parâmetros que vão ser avaliados e os respetivos indicadores (medidas utilizadas para definir se os critérios das hipóteses foram, ou não, cumpridos):

Os indicadores de avaliação são os seguintes:

1. Requisitos funcionais- Validar se os requisitos funcionais estabelecidos foram cumpridos na totalidade.
2. Usabilidade- Validar a simplicidade e facilidade de utilização do sistema, quer na perspetiva de utilizador como colaborador.
3. Fiabilidade- Garantir que todas as funcionalidades do sistema não falham e não existe possibilidade de erros.
4. Utilidade- Validar se o sistema vai de encontro às necessidades atuais e se melhora a satisfação do utilizador.

As fontes de informação retratam os métodos usados para avaliar os indicadores referidos acima - cada fonte pode avaliar um ou mais indicadores. As fontes de informação a serem utilizadas são:

- Inquérito de usabilidade dos utilizadores- Inquérito realizado aos utilizadores de uma plataforma que procuram apoio através de um serviço *help desk*. Este inquérito visa avaliar os indicadores 1,2 e 4;
- Inquérito de usabilidade dos colaboradores- Inquérito colaboradores da empresa que fornece o serviço de *help desk*. Este inquérito visa avaliar os indicadores 1,2 e 4;
- Testes de carga- Desenvolvimento de testes de carga para garantir a fiabilidade e resistência a falhas do sistema. Este inquérito visa avaliar o indicador 3;

6.3 Métodos de Avaliação

Tendo por base os indicadores e fontes de informação já definidos, é necessário definir o processo de utilização destas fontes, sendo este o método como elas vão ser aplicadas e avaliadas.

O objetivo é obter uma classificação para os indicadores apresentados na secção 6.20, numa escala intervalar, entre 0 e 5. Caso o valor de cada indicador for superior a 3, refuta-se a hipótese H0 e conclui-se que a hipótese Ha é verdadeira.

6.3.1 Inquérito de Usabilidade aos Utilizadores

Um dos objetivos principais do projeto é a melhoria na comunicação entre um utilizador de uma plataforma e um colaborador do *help desk*. Como tal, é pertinente a elaboração de um inquérito para avaliar a satisfação de um utilizador comum com a utilização do sistema desenvolvido.

Previamente ao preenchimento do inquérito, é requerido ao utilizador que utilize a plataforma (com a ajuda de um guião ou apoio presencial) para colocar uma questão que será tratada por um colaborador. Pede-se ao utilizador que acompanhe a resposta à sua questão e os passos necessários para a sua resolução.

Neste inquérito é necessária a avaliação dos seguintes pontos:

- Necessidades do utilizador são estabelecidas;
- Existência de possíveis melhorias;
- Facilidade de utilização da plataforma;
- Facilidade de obter apoio e resolução a problemas.

Estes pontos devem ser avaliados através de diversas perguntas diferentes, que podem avaliar 1 ou mais pontos.

Em relação à escolha dos inquiridos para este inquérito, o objetivo é serem escolhidos utilizadores comuns - quanto menos especializados na área de informática ou na área da plataforma adotada pelo inquérito, melhor. Através desta seleção, garante-se a avaliação da usabilidade e adaptação do utilizador o melhor possível. Pois, caso fossem selecionados utilizadores experientes na área de informática, o uso da plataforma seria maioritariamente simples.

6.3.2 Inquérito de Usabilidade aos Colaboradores

Este inquérito funciona de forma semelhante ao realizado aos utilizadores, apresentados na secção 6.3.1. No entanto, é avaliada a outra perspetiva; a dos colaboradores *help desk*. Para isto, é elaborado um inquérito para avaliar a resposta do sistema, às necessidades de um colaborador help desk e se este se adequa ou não ao processo pretendido.

Novamente, pede-se ao inquirido a utilização prévia do sistema para prosseguir à resposta do inquérito (com a ajuda de um guião ou apoio presencial). Como tal, é necessário que este siga os diversos fluxos e funcionalidades do sistema (caso seja necessário, pode também ser simulado um utilizador que necessita de apoio).

Neste inquérito é necessária a avaliação dos seguintes pontos:

- Necessidades de *help desk* são estabelecidas e fiáveis;
- Existência de possíveis melhorias;
- Melhoramento do processo de *help desk*.

Estes pontos devem ser avaliados através de diversas perguntas diferentes, que podem avaliar um ou mais pontos.

Para a seleção dos inquiridos existem os seguintes requisitos:

1. Colaborador atual de *help desk*;
2. Desempenha funções de suporte na empresa atual;
3. Antigo colaborador de *help desk;*
4. Desempenhou funções de suporte;
5. Experiência técnica.

Os critérios estão ordenados por importância, apesar de esta não se refletir na avaliação final. Quanto mais alto na lista seja o estado do inquirido, mais relevante é para responder ao inquérito - sendo que o ideal seria um colaborador atual de help desk.

6.3.3 Testes de carga

Para ser possível a avaliação da fiabilidade do sistema é necessário garantir que este é resistente a uma grande quantidade de dados e intervenções. Para tal, é necessário o teste de carga das funcionalidades principais do sistema, efetuando várias operações concorrentes e garantindo que o sistema não falha.

Devido a ser um teste com uma especialidade técnica, será executada unicamente pela equipa de desenvolvimento do sistema. Existem várias ferramentas para a realização destes testes, que serão analisadas aquando das restantes decisões arquiteturais e tecnológicas.

6.4 Resultados

6.4.1 Inquérito de Usabilidade aos Clientes

Neste Inquérito demonstrado no Anexo C Anexo B , pretende-se avaliar a utilização da plataforma na perspetiva de um cliente. Para isso, é pedido a quem vai responder, para usá-la num fluxo de criação e resposta a um ticket. Neste caso, o cliente deve seguir os seguintes passos.

1. Criar um ticket;
2. Verificar que ticket aparece na listagem e que foi atribuído a um colaborador;

3. Abrir o ticket e utilizar o chat para responder ao colaborador;
4. No final da interação e da resolução do ticket verificar que este já não aparece na listagem (a menos que se filtre pelos tickets fechados).

A este questionário responderam 12 pessoas.

Em primeiro lugar, é importante definir a experiência que a maior parte dos questionados possui com aplicações informáticas, pois este valor pode influenciar o resto das respostas, no sentido que podem vir de alguém com mais experiência e, por isso, mais crítico ou alguém com pouca experiência, que pode não perceber tão bem a plataforma ou interpretar esta como sendo melhor do que realmente é.

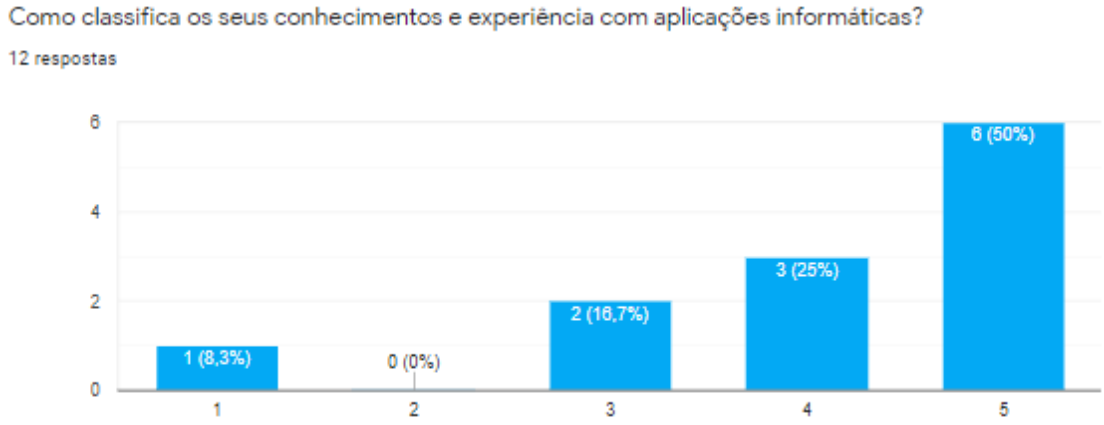


Figura 46 – Experiência dos questionados Clientes

É possível concluir, a partir da Figura 46, que a maioria dos questionados tem bastante experiência com plataformas informáticas visto que 75% selecionaram 4 ou 5.

De seguida, é realizada uma pergunta para perceber se os questionados já passaram por uma situação semelhante à do cliente da plataforma, na sua experiência pessoal. Alguém que já tenha passado por uma situação semelhante consegue perceber o que poderia ter sido melhor ou pior, e comparar a plataforma que está a utilizar com essa experiência. A esta pergunta, a maioria dos questionados (11 ou 91.7%) responderam que sim. Um resultado tão grande era esperado, visto que é comum alguém já ter passado por uma situação em que foi necessário entrar em contacto com uma empresa, para esta ação (por exemplo bancos ou companhia de eletricidade).

A terceira pergunta pretende que o questionado compare a experiência que teve na situação da pergunta anterior, com a que acabou de ter ao utilizar a plataforma.

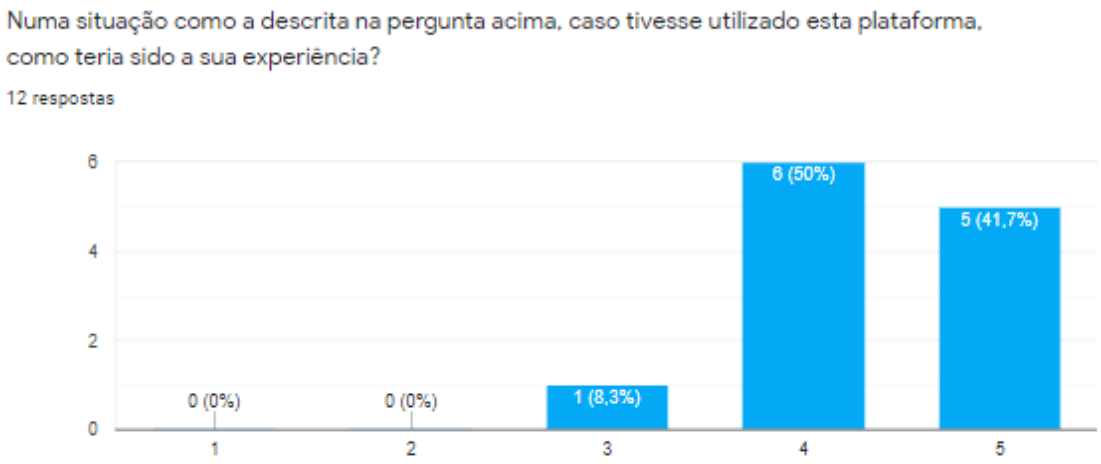


Figura 47 - Comparação de Experiência da plataforma de clientes com situação real

O resultado desta pergunta foi bastante positivo, visto que 11 pessoas (91.7%) responderam acima de 3, o que indica que a experiência foi melhor do que a que tiveram pessoalmente. No fluxo seguido pelos questionados, o chat era imediatamente respondido e o ticket foi imediatamente abordado - algo que pode ser raro numa situação real - o que poderia justificar um resultado melhor que o esperado.

As duas perguntas seguintes pretendem avaliar a plataforma de uma perspetiva de performance e estética, pelo que:

- 50% (6) dos questionados responderam 5 à performance e os restantes responderam 4. O resultado foi bastante positivo com 100% dos questionados a avaliarem a performance como positiva;
- Apenas 2 (16.7%) questionados avaliaram a interface gráfica com 3; 6 (50%) avaliaram com 4 e os restantes 4 questionados (33.3%) deram nota 5. Este resultado é positivo, mas existe definitivamente espaço para melhoras.

Seguidamente é perguntado se, caso os questionados fossem donos de um negócio, iriam gostar que os seus clientes utilizassem esta plataforma como via de suporte, ao que 10 (83.3%) responderam que sim e os restantes 2 (16.7%) que não.

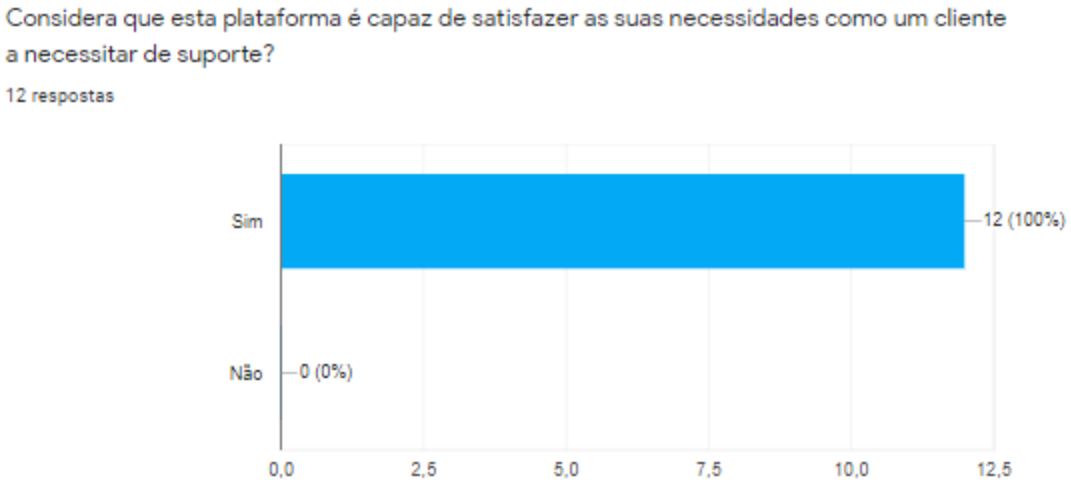


Figura 48- Questão sobre plataforma estabelecer as necessidades

A pergunta da Figura 48 representa uma das mais relevantes. Nesta pergunta, pretende-se saber se a plataforma cumpre todas as necessidades de um cliente. Todos os inquiridos (100%) responderam que sim. Isto indica que, apesar de não ser perfeita, a plataforma cumpre com o que é pretendido e requisitado.

As seguintes perguntas pretendem descobrir a avaliação geral da plataforma e se esta poderia ser recomendada, ao que:

- 58.3% (7) questionados avaliaram a plataforma com 4 e os restantes 41.7% (5) com 5;
- 100% recomendaria esta plataforma.

Apesar de existir uma margem para melhorias, estes resultados comprovam que a plataforma cumpre os requisitos e que os questionados gostaram da dela e gostariam de a utilizar caso precisassem de pedir suporte a uma empresa.

6.4.2 Inquérito de Usabilidade aos Clientes

Neste Inquérito, demonstrado no Anexo C Anexo A , pretende-se avaliar a utilização da plataforma, como um colaborador. Para isso, é pedido a quem vai responder, para usá-la num fluxo de resposta a um ticket. Neste caso, o colaborador deve seguir os seguintes passos:

5. Ver listagem de tickets;
6. Verificar que existe um ticket atribuído e abrir o mesmo;
7. Verificar histórico do ticket;
8. Alterar a prioridade do ticket;
9. Iniciar a conversa com o cliente até resolver o ticket;

10. Adicionar uma nota no ticket sobre a resolução;
11. Alterar o estado do ticket para fechado e guardar as alterações;
12. Verificar que o ticket já não existe na listagem (a menos que se filtre pelos tickets fechados).

Este questionário obteve um total de 14 respostas.

Tal como na secção anterior, primeiramente é importante perceber o nível de experiência dos questionados com plataformas informáticas. Esta primeira questão teve um resultado bastante disperso, sendo que a maioria respondeu com 5:

- 1 questionado (7.1%) respondeu 1;
- 2 questionados (14%) responderam 3;
- 3 questionados (21.4%) responderam 4;
- 8 questionados (51.7%) responderam 5;

Na pergunta seguinte, é questionado se o inquirido já esteve numa situação em que foi necessário suporte a um cliente. A esta pergunta, 57.1% (8) respondeu que sim. Este resultado é maior que o esperado, no entanto, devido a existir um grande número de questionados com formação numa área relacionada com tecnologia, é possível justificar este resultado com o facto de possivelmente já terem passado por essa situação, na sua vida profissional.

Em seguida, é novamente pedido aos questionados para compararem a sua experiência ao utilizar a plataforma, com o que experienciaram a nível pessoal.

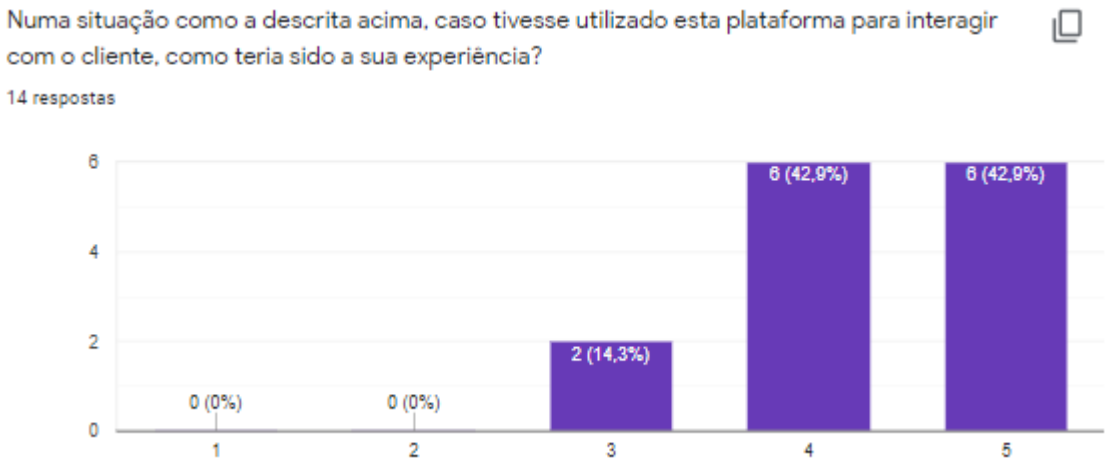


Figura 49 - Comparação de experiência real com plataforma *help desk*

Os resultados da Figura 49 permitem a conclusão de que a experiência na plataforma foi positiva, comparada com a que os questionados tiveram.

As três perguntas seguintes pretendem que o questionado avalie a plataforma, na perspetiva de simples utilização - performance e estética. A estas perguntas foram obtidos os seguintes resultados:

- 42.9% (6) avaliaram a utilização da plataforma com 4, os restantes 57.1% (8) com 5;
- 78.6% (11) avaliaram a performance com 5, os restantes 21.4% (3) com 4;
- Apenas 14.3% (2) avaliaram a interface gráfica da plataforma com 3, 42.9% (6) avaliaram com 4 e os restantes 42.9% (6) com 5;

Estes resultados são positivos, embora indiquem que podem existir melhorias, principalmente na interface gráfica.

A seguinte pergunta pretende saber se o questionado utilizaria a plataforma, no seu próprio negócio (caso o tivesse) ao que 92.9% (13) respondeu que sim.

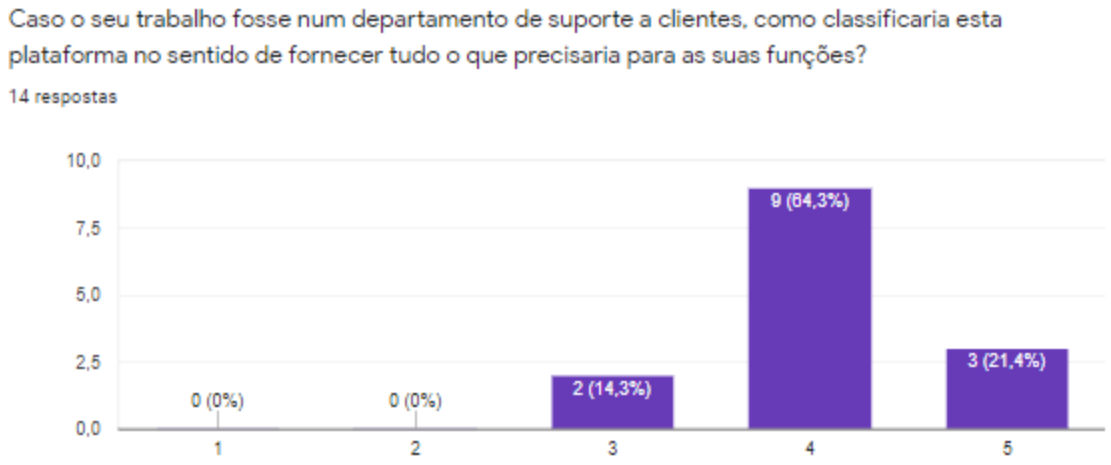


Figura 50 - Classificação da plataforma para sistema de *help desk*

A pergunta da Figura 50 pretende perceber se os questionados gostariam de utilizar esta plataforma, caso trabalhassem para um departamento de suporte a clientes. Os resultados foram bastante positivos, sem existir nenhum negativo.

A última pergunta é se o questionado recomendaria esta plataforma, ao que 100% respondeu que sim.

Estes resultados são muito positivos e permitem concluir que, apesar de existirem possíveis melhorias, os questionados ficaram satisfeitos com a plataforma e gostariam de a utilizar caso estivessem nessa situação.

6.4.3 Testes de Carga

Os testes de carga são utilizados para testar a capacidade do sistema de responder eficientemente enquanto é sobrecarregado com pedidos. O sistema tem de ser capaz de funcionar corretamente nesta situação.

Para estes testes é primeiramente necessário definir quais funcionalidades testar e o que esperar destes testes. As funcionalidades ideais foram escolhidas através dos seguintes parâmetros:

- Complexidade;
- Importância para o sistema;
- Utilização;

Sendo assim, foram definidas para efetuar testes de carga:

- Criar um ticket (*endpoint*: "/api/ticket"), esta é uma das funcionalidades principais do sistema e a que envolve mais lógica e passos, como pode ser observado no diagrama de sequência da Figura 17. Devido a isto, este é um dos *endpoints* mais propícios a ter problemas quando sobrecarregado;
- Enviar mensagem (*endpoint*: "/api/ticket/{ticket_id}/chatMessage"), esta funcionalidade foi escolhida por potencialmente ser uma das mais utilizadas, caso existam vários tickets em aberto, podem estar constantemente a ser escritas mensagens nestes;

Para efetuar os testes de carga foi utilizada a plataforma "Apache JMeter", uma plataforma desenvolvida com o âmbito de realizar testes de carga (entre outros). Na Figura 51 é possível observar as configurações para os testes do "JMeter" (do lado esquerdo), assim como um gráfico onde são apresentados os resultados de um dos testes efetuados:

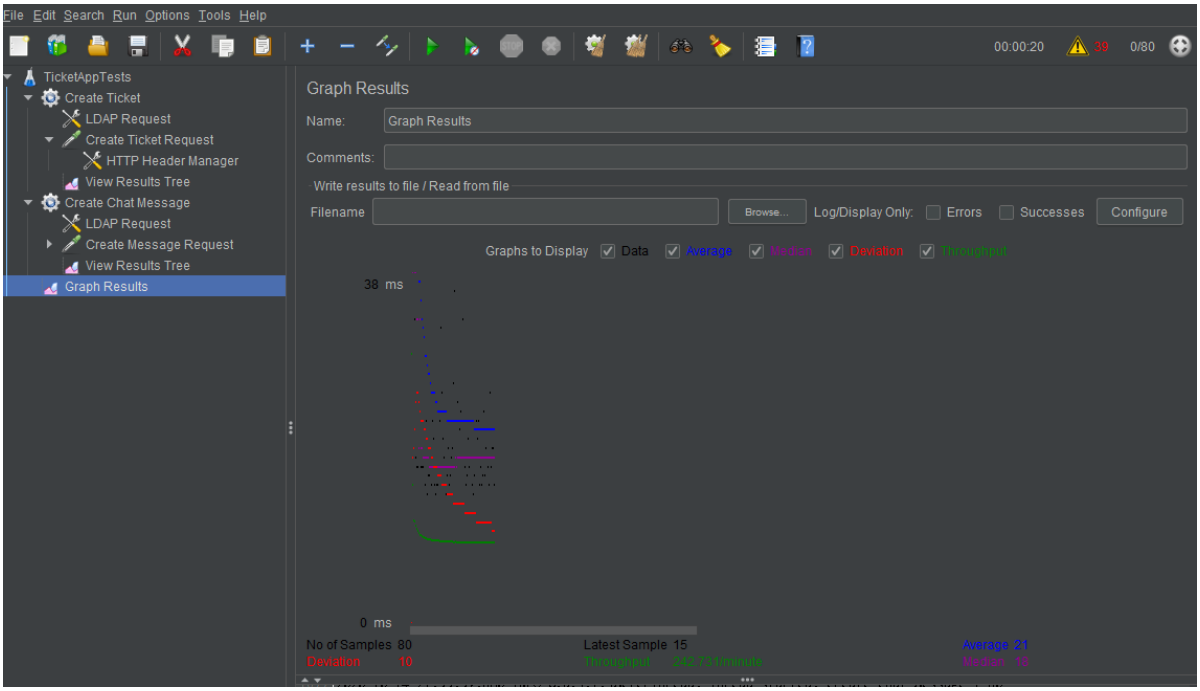


Figura 51 - Configuração "JMeter"

De seguida são definidos os testes a serem efetuados, para cada uma das funcionalidades são realizados 3 testes:

1. Duração: 20 segundos; Pedidos por segundo: 2
2. Duração: 20 segundos; Pedidos por segundo: 4
3. Duração: 20 segundos; Pedidos por segundo: 10

Nestes 3 testes são avaliados 3 componentes principais:

- *Throughput*: Número de pedidos que o sistema aguenta por minuto sobre estas condições, numa situação ideal, número será aproximadamente o número de pedidos efetuados nos 20 segundos multiplicado por 4 (por exemplo, o teste 2 tem 80 pedidos em 20 segundos (4 por segundo), por isso o *throughput* deverá ser 240);
- D*eviation:* Este valor representa o desvio da média ideal, ou seja, no exemplo acima, quanto mais o *throughput* seja diferente do que 240, maior será a *deviation*, a *deviation* ideal é a mais próxima de 0 possível.
- Número de Erros: Nos 2 itens anteriores avaliamos a performance do sistema numa situação de sobrecarga, nesta avaliamos a sua resiliência. Caso existam erros significa que a sobrecarga levou o sistema a falhar;

Visto estes testes estarem a ser realizados com os serviços a correr num ambiente local, estes podem ser influenciados, devido a isso é importante considerar que, num ambiente de produção (ou de QA (exemplo de um ambiente especifico para testes)), os testes teriam melhores resultados e é algo importante a ser testado quando este ambiente existir.

6.4.3.1 Resultados Criação de Ticket

Os resultados para os 3 testes a esta funcionalidade são representados na Tabela 16:

Tabela 16 - Resultados dos testes de carga a Criação de Tickets

Teste	Throughput	Deviation	Erros
1	122.33	46	0
2	214.832	1195	0
3	185.191	9638	3

Como era esperado o primeiro teste teve resultados positivos, no entanto no segundo já foram encontrados problemas, o *throughput* está um pouco espaçado do ideal (240), e a *deviation* tem um valor consideravelmente alto. Os resultados do teste 2 anteveem o que vai acontecer no teste 3, neste teste temos resultados negativos, a *deviation* toma valores bastante grandes e em 3 dos pedidos a aplicação falhou.

Com estes resultados é possível concluir que existem problemas de performance nesta funcionalidade que devem ser considerados para trabalho futuro (relembra-se que num ambiente de produção ou de *qa* estes resultados melhorar). No entanto, adaptando a uma situação real, esta funcionalidade não vai ser utilizada frequentemente, possivelmente nunca chegará a uma situação como a do teste 1, mesmo assim é algo a ser melhorado.

6.4.3.2 Resultados envio de mensagem

Os resultados dos testes de carga para esta funcionalidade podem ser observados na Tabela 17:

Tabela 17- Resultados dos testes de carga a Envio de mensagem

Teste	Throughput	Deviation	Erros
1	122.511	21	0
2	242.731	10	0
3	431.499	2307	0

Tal como esperado, devido a ser uma funcionalidade com poucos passos (a publicação no pusher (o único passo adicional desta funcionalidade) é realizada de forma assíncrona), os testes tiveram resultados positivos, apenas o terceiro teste tem uma *deviation* um pouco alta o que indica alguma perda de performance, no entanto não houve erros nos pedidos.

7 Conclusão

A elaboração desta dissertação teve como principal objetivo, o desenvolvimento de um sistema de *help desk* capaz de melhorar a eficiência e eficácia do processo de suporte que uma empresa oferece aos seus clientes.

Primeiramente, foi elaborado um pequeno capítulo de introdução para apresentar e contextualizar o problema e definir os objetivos principais a concretizar.

De seguida, foi realizada uma análise do estado da arte atual. Neste capítulo, o problema foi contextualizado com maior detalhe e foram analisadas soluções já existentes no mercado e as suas qualidades e limitações. Por fim, foram analisadas algumas tecnologias que necessitavam de uma tomada de decisão, sobre qual adotar. Com o resultado desta análise, foi realizada uma análise de valor à aplicação. Aqui foi feita uma análise de inovação à plataforma a desenvolver e definidos alguns requisitos principais.

Seguidamente, foi realizada a Análise e Design da plataforma a desenvolver. Esta fase é das mais importantes no processo de desenvolvimento e consiste em planear e desenhar a solução a desenvolver. Aqui foram definidos os requisitos e conceitos de negócio, assim como desenhado o desenvolvimento destes requisitos.

Finalmente, a solução foi implementada. Neste capítulo é descrito o desenvolvimento do sistema, assim como alguns obstáculos encontrados e as suas soluções. Após a solução estar implementada, é necessária a sua avaliação. Nesta fase foram realizados dois inquéritos de satisfação, a utilizadores de ambas as plataformas. Estes inquéritos demonstraram um grau de satisfação médio/alto.

Conclui-se, assim, que o resultado deste projeto é positivo. A solução idealizada foi implementada e está pronta para ser escalada e utilizada. De seguida, são descritos os objetivos alcançados, assim como as limitações e trabalho futuro.

7.1 Objetivos alcançados

Os objetivos do sistema a desenvolver passavam por construir uma solução apta a melhorar o processo de suporte de uma empresa, para os seus clientes. Esta solução deveria ser capaz de ser integrada em qualquer negócio ou plataforma e poder ser escalada com novas funcionalidades, quando necessário.

Na secção 0 foram definidas algumas funcionalidades principais para este sistema, sendo estes:

Tabela 18- Resultado dos objetivos

Objetivo	Resultado
1- Criação de tickets e gestão de tickets previamente criados	Atingido
2- Visualização de alterações	Atingido
3- Chat entre o Colaborador e o Utilizador	Atingido
4- Notificações a utilizadores aquando de alterações no ticket	Não Atingido
5- Distribuição automática de tickets	Atingido
6- Possibilidade de reatribuição de tickets	Atingido
7- Visão de análise administrativa	Não Atingido
8- Integrabilidade em qualquer negócio ou plataforma	Atingido

Como é possível observar, apenas 2 objetivos não foram atingidos. Em relação ao objetivo 4, o sistema está preparado para a implementação do serviço para as notificações. Este serviço é apresentado na Figura 13 como "EmailNotificationService", no entanto, não chegou a ser implementado.

Em relação ao objetivo 7, este é representado como o caso de uso UC04, no entanto, pelo mesmo motivo referido acima, este caso de uso não foi implementado.

7.2 Limitações e trabalho futuro

Apesar do esforço de desenvolvimento para atingir todos os objetivos e ultrapassar os obstáculos encontrados, não foi possível atingir todas as necessidades. Na verdade, existe sempre algo a acrescentar a uma solução pelo que esta nunca está verdadeiramente concluída.

Visto isto, foram encontradas algumas limitações que não foram ultrapassadas, além destas existem outros requisitos e funcionalidades que poderiam melhorar a solução e que, como tal, vale a pena mencionar como trabalho futuro:

1. Finalização dos objetivos estabelecidos: Na secção 7.1 foram mencionados alguns objetivos que não foram cumpridos, é importante que no futuro estes sejam implementados;
2. Pipelines: Devido a limitações de recursos, apenas foram implementadas pipelines de CI e CD para o serviço "TicketInfoService", estas pipelines são importantes pois ajudam

a automatizar os processos de teste e publicação do sistema, por isso seria importante desenvolver para os restantes serviços. Mesmo a pipeline desenvolvida para o "TicketInfoService" não foi desenvolvida para ser publicada através de "containers", deve ser analisado e implementado para coincidir com o planeado;

3. Melhorar o processo de automação aquando da criação e atribuição de tickets: Podem ser adotados sistemas de *chatbots* para realizar o primeiro contacto com o Utilizador, o sistema também pode ser capaz de, através dos campos descritos no ticket, perceber o melhor departamento a responder ao mesmo e automaticamente atribuir o ticket a esse departamento. Estes são apenas exemplos de possíveis melhorias para este requisito, no entanto deve ser realizada uma análise mais aprofundada para perceber a melhor alternativa;
4. Biblioteca de testes: Para todos os serviços foram apenas desenvolvidos testes unitários e integração, sendo que os de integração apenas testam as camadas de lógica e persistência de dados (poderiam também testar a camada de Apresentação). Devem ser desenvolvidos mais testes como Testes de Aceitação e Testes *End-To-End* para melhorar a confiabilidade do sistema;
5. Analisar e Desenvolver novos requisitos: Como foi apresentado na secção , existem várias soluções para resolver o mesmo problema deste sistema, como tal é necessário que este sistema esteja sempre num processo de melhoria constante para se manter a par com as restantes plataformas, para isto é necessária uma análise de possíveis novos requisitos a implementar, como o referido no ponto 3;

Referências

ACM. (s.d.). Obtido de ACM: https://dl.acm.org/

ActiveWizards. (s.d.). *A Comparative Analysis of ChatBots APIs.* Obtido de ActiveWizards: https://activewizards.com/blog/a-comparative-analysis-of-chatbots-apis/#H7

Andreas Goeb, K. L. (Setembro de 2011). A Software Quality Model for SOA. *A Software Quality Model for SOA*. Obtido de https://dl.acm.org/doi/abs/10.1145/2024587.2024593

Apache Kafka - Cluster Architecture. (s.d.). Obtido de Tutorials Point: https://www.tutorialspoint.com/apache_kafka/apache_kafka_cluster_architecture.htm

Brown, R. (2015). *What is a Help Desk and Its Importance for Your Organization*. Obtido de Invensis.

Capterra. (s.d.). *Help Desk Ticketing Systems*. Obtido de Capterra: https://www.capterra.com/sem-compare/help-desk-software?gclid=CjwKCAiAhJTyBRAvEiwAln2qB9rz6lEs9VeVmuUGxf0TjnPisaBIUwVRvBGJS-hJrh8s7jLxg416qxoCrKEQAvD_BwE

Crawford, C., CA, & Benedetto, C. (2005). New Products Management. 8th edition ed. Home-.

Cruess, A. (Novembro de 2002). Transforming a Help Desk from Average to Excellent. *Transforming a Help Desk from Average to Excellent*. Obtido de https://dl.acm.org/doi/abs/10.1145/588646.588702

Dewulf, K. (03 de 2013). Sustainable Product Innovation: The Importance of the Front-End Stage in the Innovation Process. *Sustainable Product Innovation: The Importance of the Front- End Stage in the Innovation Process*, pp. 139-166.

Eeles, P. (01 de 06 de 2004). IBM. *What, no supplementary specification?* Obtido de https://www.ibm.com/developerworks/rational/library/3975.html

Evans, K., & Teresa Jones, W. (2005). Building an IT Help Desk—From Zero to Hero. *Building an IT Help Desk—From Zero to Hero*, pp. 68-74.

Fowler, M. (08 de 1 de 2019). *Software Architecture Guide*. Obtido de Martin Fowler: https://www.martinfowler.com/architecture/

Freshworks. (s.d.). *Freshdesk*. Obtido de Freshdesk help desk: https://freshdesk.com/help-desk-software

Freshworks. (s.d.). *Freshdesk Features*. Obtido de Freshdesk: https://freshdesk.com/features

Google Scholar. (s.d.). Obtido de https://scholar.google.com/

Guozhang Wang, Joel Koshy, Sriram Subramanian, Kartik Paramasivam, Mammad Zadeh, Neha Narkhede, . . . Joe Stein. (Agosto de 2015). Building a Replicated Logging System with Apache Kafka. *Building a Replicated Logging System with Apache Kafka*. Obtido de https://dl.acm.org/doi/abs/10.14778/2824032.2824063

Hagberg, J., Sundstrom, M., & Egels-Zandén, N. (2016). The digitalization of retailing: An exploratory framework. *The digitalization of retailing: An exploratory framework*, pp. 694-712.

Haq, S. u. (2 de Maio de 2018). *Introduction to Monolithic Architecture and MicroServices Architecture*. Obtido de Medium: https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63

Hat, R. (s.d.). *What is event-driven architecture?* Obtido de Red Hat: https://www.redhat.com/en/topics/integration/what-is-event-driven-architecture

Humphrey, P. (26 de 04 de 2017). *Understanding When to use RabbitMQ or Apache Kafka*. Obtido de Pivotal: https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka

IdentityServer4. (2016). *Reference Tokens*. Obtido de Identity Server: https://docs.identityserver.io/en/release/topics/reference_tokens.html

Jonas DeMuro, B. T. (2 de Dezembro de 2019). *Best helpdesk software of 2020: for ticketing and support*. Obtido de Techradar: https://www.techradar.com/best/best-helpdesk-software

Koen, P., Ajamian, G., Burkart, R., Clamen, A., Davidson, J., & D'Amore, R. (2001). Providing clarity and a common language to the "fuzzy front end". Research-Technology Management.

Limitation of monolithic architecture versus its solution with microservices. (2020). Obtido de Packtub: https://subscription.packtpub.com/book/application_development/9781787281448/1/ch01lvl1sec11/limitation-of-monolithic-architecture-versus-its-solution-with-microservices

Mansfield, D. (2018). *Marketing theory: understanding customer value*. Obtido de BuiltVisible: https://builtvisible.com/understanding-customer-value/

Microservices Architecture Design and Best Practices. (2018). Obtido de XenonStack: https://www.xenonstack.com/insights/microservices/

Microservices architecture style. (s.d.). Obtido de Microsoft: https://martinfowler.com/articles/microservices.html

Morais, M. (2018). *As principais diferenças de comunicar para B2C e B2B.* Obtido de OutMarketing: https://outmarketing.pt/blog/as-principais-diferencas-de-comunicar-para-b2c-e-b2b/

Packter, B. (2 de 12 de 2018). *Kafka vs RabbitMQ*. Obtido de ITNext: https://itnext.io/kafka-vs-rabbitmq-f5abc02e3912

Patterson, J. (Abril de 2017). *8 Features to Look for in a Help Desk Ticketing System.* Obtido de Transcosmos: http://transcosmos.co.uk/blog/features-look-for-help-desk-ticketing-system-2/

Pinard, D., Evans, R., & Mankovskii, S. (2001). Web based help desk. p. 10. Obtido de https://patents.google.com/patent/US6230287B1/en

Rehan, A. (2019). *9 Best Chatbot Development Frameworks to Build Powerful Bots.* Obtido de Geekflare: https://geekflare.com/chatbot-development-frameworks/

Richardson, C. (2019). *Pattern: Monolithic Architecture*. Obtido de Microservice Architecture: https://microservices.io/patterns/monolithic.html

Rongala, A. (25 de Maio de 2015). *What is a Help Desk and Its Importance for Your Organization*. Obtido de Invensis: https://www.invensis.net/blog/customer-service/what-is-help-desk-and-its-importance-for-your-organization/

SarthakGarg. (2020). *Service-Oriented Architecture*. Obtido de GeeksForGeeks: https://www.geeksforgeeks.org/service-oriented-architecture/

Silva, A. F. (2020). *O QUE É BUSINESS MODEL CANVAS E COMO FAZER UM?* Obtido de Guia Emprendedor: https://guiaempreendedor.com/o-que-business-model-canvas/

Sperl, G. (2006). Taming the Help Desk. *Taming the Help Desk*.

Techopedia. (s.d.). *Help Desk*. Obtido de Techopedia: https://www.techopedia.com/definition/353/help-desk

Tosca, C. (2018). *What is Value Analysis / Value Engineering*. Obtido de Bruschi: https://www.bruschitech.com/blog/what-is-value-analysis-value-engineering

Watts, S. (31 de Maio de 2017). *What is SOA? Service-Oriented Architecture Explained*. Obtido de BMC: https://www.bmc.com/blogs/service-oriented-architecture-overview/

Woodall, T. (2003). Conceptualising 'Value for the Customer': An Attributional, Structural and Dispositional Analysis.

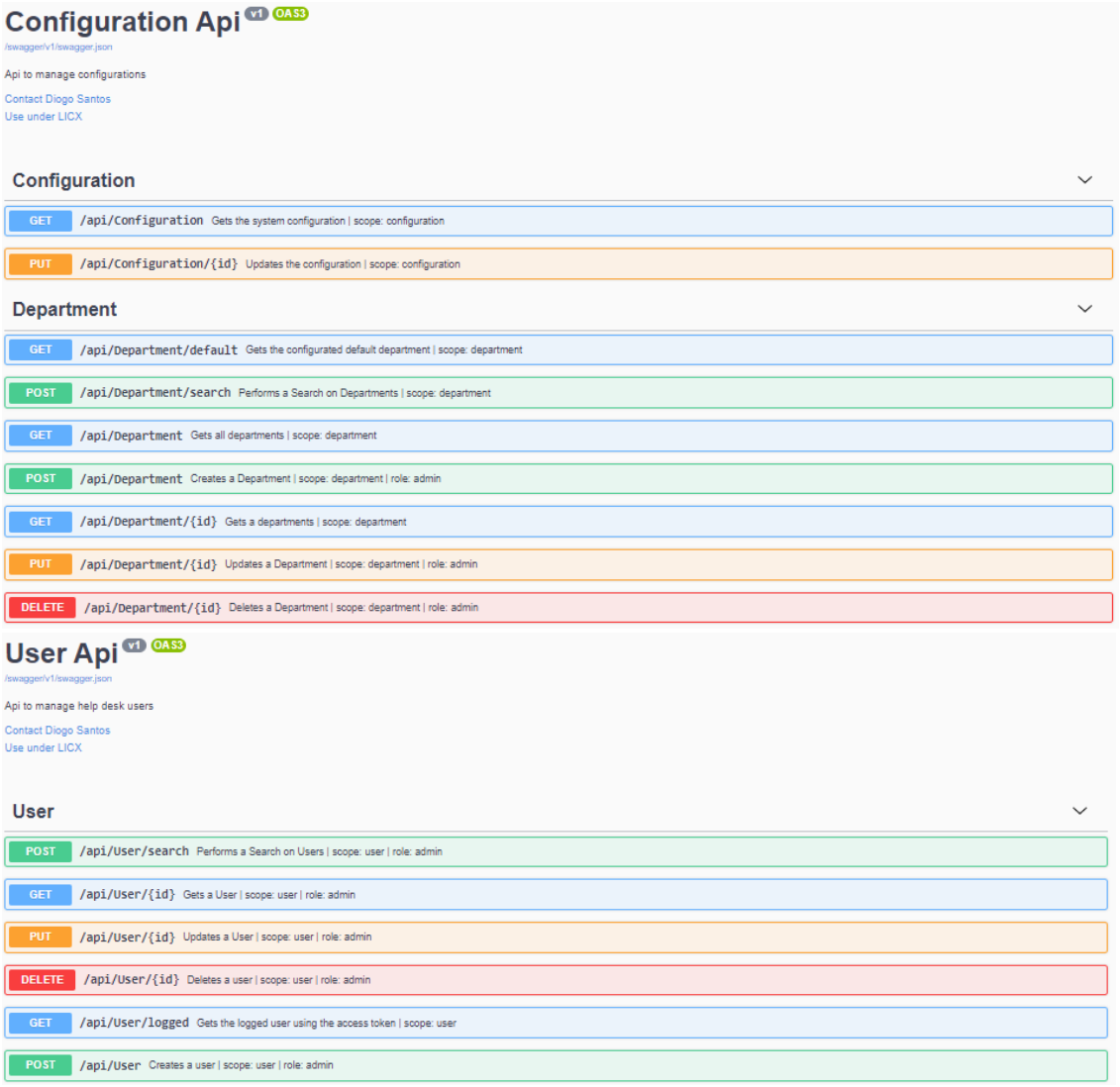
Zhongshan Ren, W. W. (Setembro de 2018). Migrating Web Applications from Monolithic Structure to Microservices Architecture. *Migrating Web Applications from Monolithic Structure to Microservices Architecture*. Obtido de https://dl.acm.org/doi/abs/10.1145/3275219.3275230

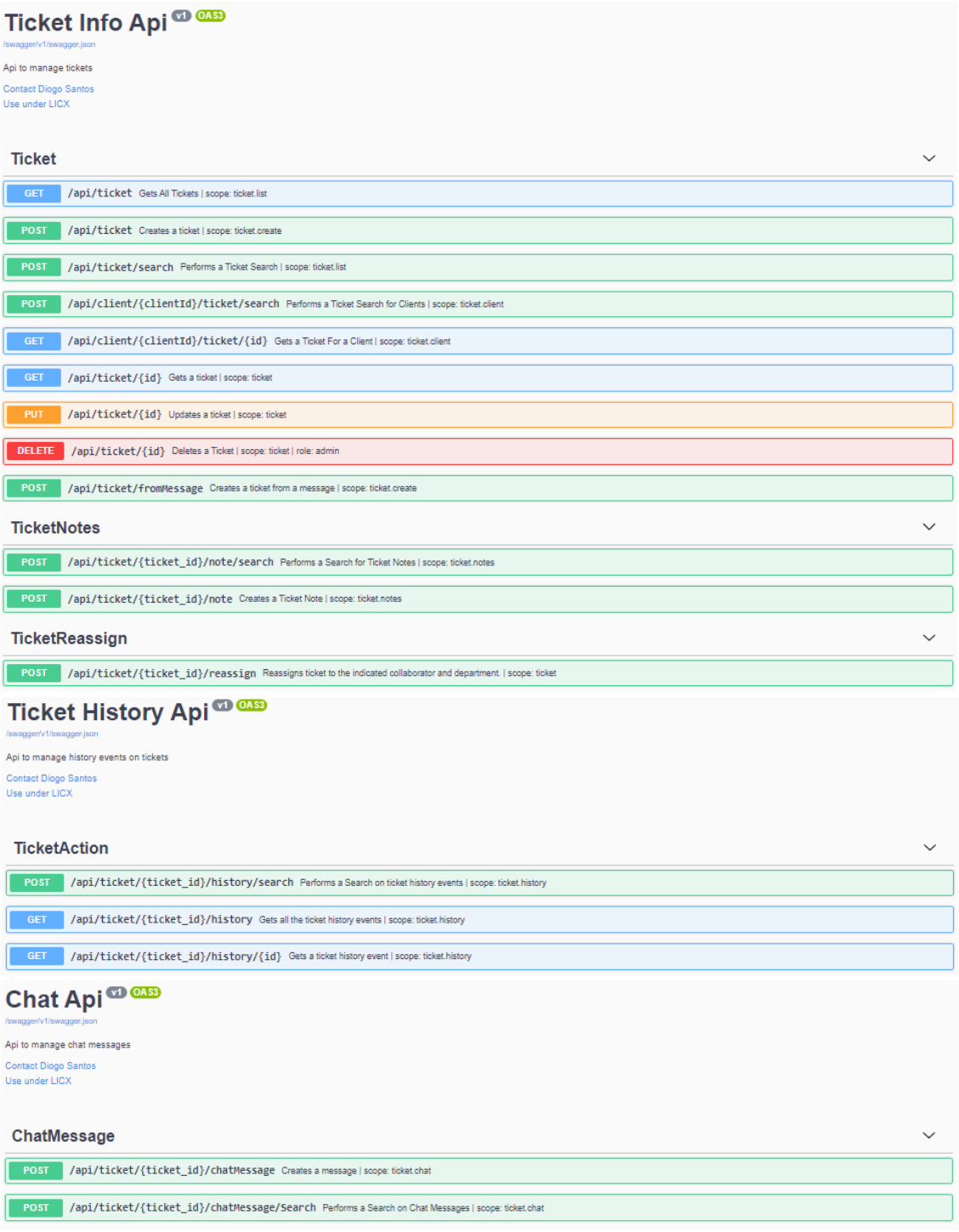
Zoho. (s.d.). *Zoho desk*. Obtido de Zoho desk: https://www.zoho.com/desk/features.html

Zoho. (s.d.). *Zoho Desk*. Obtido de Zoho: https://www.zoho.com/desk/features.html

Anexo A *Endpoints* dos Serviços

Neste anexo são representados os vários *endpoints* dos serviços, através da ferramenta *Swagger*, na descrição dos mesmos foi também adicionados as restrições de acesso (Autorização).




Ticket Info Api
v1
OAS3
/swagger/v1/swagger.json
Api to manage tickets
Contact Diogo Santos
Use under LICX
Ticket
GET /api/ticket Gets All Tickets | scope: ticket.list
POST /api/ticket Creates a ticket | scope: ticket.create
POST /api/ticket/search Performs a Ticket Search | scope: ticket.list
POST /api/client/{clientId}/ticket/search Performs a Ticket Search for Clients | scope: ticket.client
GET /api/client/{clientId}/ticket/{id} Gets a Ticket For a Client | scope: ticket.client
GET /api/ticket/{id} Gets a ticket | scope: ticket
PUT /api/ticket/{id} Updates a ticket | scope: ticket
DELETE /api/ticket/{id} Deletes a Ticket | scope: ticket | role: admin
POST /api/ticket/fromMessage Creates a ticket from a message | scope: ticket.create
TicketNotes
POST /api/ticket/{ticket_id}/note/search Performs a Search for Ticket Notes | scope: ticket.notes
POST /api/ticket/{ticket_id}/note Creates a Ticket Note | scope: ticket.notes
TicketReassign
POST /api/ticket/{ticket_id}/reassign Reassigns ticket to the indicated collaborator and department. | scope: ticket
Ticket History Api
v1
OAS3
/swagger/v1/swagger.json
Api to manage history events on tickets
Contact Diogo Santos
Use under LICX
TicketAction
POST /api/ticket/{ticket_id}/history/search Performs a Search on ticket history events | scope: ticket.history
GET /api/ticket/{ticket_id}/history Gets all the ticket history events | scope: ticket.history
GET /api/ticket/{ticket_id}/history/{id} Gets a ticket history event | scope: ticket.history
Chat Api
v1
OAS3
/swagger/v1/swagger.json
Api to manage chat messages
Contact Diogo Santos
Use under LICX
ChatMessage
POST /api/ticket/{ticket_id}/chatMessage Creates a message | scope: ticket.chat
POST /api/ticket/{ticket_id}/chatMessage/Search Performs a Search on Chat Messages | scope: ticket.chat

Anexo B Questionário Clientes

Formulário Plataforma HelpDesk (Clientes)

Este questionário visa avaliar a plataforma de utilizadores do sistema help desk desenvolvido. Antes de responder pede-se que corra a plataforma como um utilizador pedindo suporte, passando por todos os passos necessários como: Criar ticket, ver listagem de tickets e utilizar o chat para falar com o colaborador.
Após este fluxo deve ser capaz de avaliar a plataforma respondendo a este questionário, Obrigado.

*Obrigatório

Como classifica os seus conhecimentos e experiência com aplicações informáticas? *

	1	2	3	4	5	
Não estou habituado/a e tenho dificuldades	◯	◯	◯	◯	◯	Tenho formação numa área relacionada

Já alguma vez se encontrou numa situação em que foi necessário pedir suporte ou ligar para a empresa para apoio (ou questões) com algo relacionado com o produto ou serviço deles? *

- ☐ Sim
- ☐ Não

Numa situação como a descrita na pergunta acima, caso tivesse utilizado esta plataforma, como teria sido a sua experiência? *

	1	2	3	4	5	
Má	◯	◯	◯	◯	◯	Excelente

Como avalia esta plataforma no sentido de performance (rapidez de resposta a cada ação) *

	1	2	3	4	5	
Muito lenta	◯	◯	◯	◯	◯	Muito rápida

Como avalia a interface gráfica da plataforma? *

	1	2	3	4	5	
Má	○	○	○	○	○	Excelente

Caso tivesse um negócio, gostava que os seus clientes tivessem acesso a esta plataforma? *

- [ ] Sim
- [ ] Não

Considera que esta plataforma é capaz de satisfazer as suas necessidades como um cliente a necessitar de suporte? *

- [ ] Sim
- [ ] Não

Como classifica esta plataforma no geral? *

	1	2	3	4	5	
Má	○	○	○	○	○	Excelente

Recomendaria esta plataforma aos seus amigos? *

- [ ] Sim
- [ ] Não

Existe alguma nota que queira acrescentar sobre a plataforma?

A sua resposta

Submeter

Anexo C Questionário Colaboradores

Questionário Plataforma Helpdesk (Colaboradores)

Este questionário visa avaliar a plataforma de colaboradores do sistema help desk desenvolvido. Antes de responder pede-se que corra a plataforma como um colaborador respondendo ao pedido de ajuda de um cliente, passando por todos os passos necessários como: Ver ticket na listagem, ver histórico do ticket, enviar mensagem de chat, criar uma nota e por último alterar o estado do ticket para fechado.
Após este fluxo deve ser capaz de avaliar a plataforma respondendo a este questionário, Obrigado.

*Obrigatório

Como classifica os seus conhecimentos e experiência com aplicações informáticas. *

	1	2	3	4	5	
Não estou habituado/a e tenho dificuldades,	◯	◯	◯	◯	◯	Tenho formação numa área relacionada

Já alguma vez teve numa situação em que foi preciso dar suporte ou ajuda a um cliente? *

- ☐ Sim
- ☐ Não

Numa situação como a descrita acima, caso tivesse utilizado esta plataforma para interagir com o cliente, como teria sido a sua experiência? *

	1	2	3	4	5	
Má	◯	◯	◯	◯	◯	Excelente

Como avalia esta plataforma no sentido de intuitividade (simples utilização). *

	1	2	3	4	5	
Má	◯	◯	◯	◯	◯	Excelente

Como avalia esta plataforma no sentido de performance (velocidade de resposta). *

	1	2	3	4	5	
Má	○	○	○	○	○	Excelente

Como avalia a interface gráfica da plataforma? *

	1	2	3	4	5	
Má	○	○	○	○	○	Excelente

Caso tivesse um negócio, adoptava esta plataforma para oferecer suporte aos seus clientes? *

- [ ] Sim
- [ ] Não

Caso o seu trabalho fosse num departamento de suporte a clientes, como classificaria esta plataforma no sentido de fornecer tudo o que precisaria para as suas funções? *

	1	2	3	4	5	
Má	○	○	○	○	○	Excelente

Recomendaria esta plataforma a alguém que precisasse de melhorar o departamento de suporte do seu negócio? *

- [ ] Sim
- [ ] Não

Tem mais alguma nota a acrescentar sobre esta plataforma?

A sua resposta

Submeter