INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Microsserviços .NET para suportar uma plataforma de e-fulfillment

ADEMAR FILIPE DA CUNHA VALE
Junho de 2023

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Microsserviços .NET para suportar uma plataforma de e-fulfillment

Ademar Filipe da Cunha Vale

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

Orientador: Nuno Ferreira

Porto, junho 2023

Declaração de Integridade

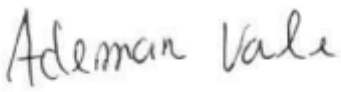
Declaro ter conduzido este trabalho académico com integridade.

Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Portanto, o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

ISEP, Porto, 30 de junho de 2023



Querido pai…

Resumo

O Spoke é uma plataforma maioritariamente direcionada ao *e-commerce* desenvolvida pela antiga start-up portuguesa HUUB. O principal objetivo desta aplicação passa por suportar todo o processo de negócio de empresas que utilizem plataformas online para venda de produtos, através de integração com diversos tipos de plataformas e sistemas.

A este mesmo processo dá-se o nome de *e-fulfillment* e este é responsável pelo processamento de uma compra online: receber, armazenar, selecionar, embalar e distribuir produtos.

Motivada pela visão e tecnologia desenvolvida pela HUUB, em 2021, a operadora logística multinacional Maersk chegou a um acordo para a compra total da primeira. Com esta compra surgiram novos desafios, não só a nível de paradigma, mas também a nível de tecnologia.

O trabalho demonstrado neste documento é realizado em contexto empresarial (Maersk) e nele é abordado o início do processo de migração do Spoke para uma nova plataforma responsável pelo *e-fulfillment* de todas as empresas que desejem este serviço por parte da Maersk.

Com esta migração é também importante considerar os constantes avanços tecnológicos, assim como ferramentas que consigam tornar o desenvolvimento mais simples e eficiente. Tendo por base este pressuposto e, tendo em conta que o Spoke é uma aplicação maioritariamente monolítica desenvolvida em Python, foi tomada a decisão de, para além de optar por uma arquitetura orientada a microsserviços devido às vantagens que este estilo arquitetural apresenta, também alterar a tecnologia nas quais os serviços estão assentes para .NET (C#).

Sendo assim, neste documento são analisados ambos os estilos arquiteturais acima referidos, estratégias de migração e boas práticas de desenvolvimento de software. Através da realização de uma análise arquitetural, são propostas diferentes arquiteturas resultantes do processo de segmentação, das quais uma é selecionada e desenvolvida tendo por base, não só os fundamentos teóricos citados, mas também os requisitos identificados.

Palavras-chave: *e-commerce*, *e-fulfillment*, microsserviços, monolítico, arquitetura, software, .NET, Python

Abstract

Spoke is an e-commerce related platform developed by the former Portuguese start-up HUUB. The main objective of this application is to support the entire business process of companies that use online platform to sell products, through integration with various types of platforms and systems.

This process is called e-fulfillment and it is responsible for the entire process related to online purchases: receiving, storing, selecting, packing, and distributing products.

Motivated by the vision and technology developed by HUUB, in 2021, the multinational logistics operator Maersk reached an agreement for the total purchase of the previous. With this acquisition came new challenges, not only at the paradigm level, but also at the technology level.

The work demonstrated in this document is done in a corporate context (Maersk) and in it is approached the beginning of the migration process of Spoke to a new platform responsible for the e-fulfillment of all companies that want this service by Maersk.

In this migration it is also important to consider the constant technological advances, as well as tools that can make the development simpler and more efficient. Based on this assumption and considering that Spoke is a mostly monolithic application developed in Python, a decision was made to not only choose a microservices oriented architecture due to the advantages that this architectural style presents, but also change the technology on which the services are based to .NET (C#).

Therefore, in this document are analysed both above mentioned architectural styles, various migration strategies and software development best practices. Through the analysis, different architectures resulting from the segmentation process are proposed, of which one is selected and developed based not only on the theoretical foundations mentioned above, but also on the requirements identified.

Keywords: e-commerce, e-fulfillment, microservices, monolithic, architecture, software, .NET, Python

x

Índice

Lista de Figuras

Lista de Tabelas

xx

Acrónimos e Símbolos

Lista de Acrónimos e símbolos

API	*Application Programming Interface*
AWS	*Amazon Web Services*
CQRS	*Command Query Responsibility Segregation*
DDD	*Domain-Driven Design*
ECL	*E-commerce Logistics*
EFF	*E-fulfillment*
ERP	*Enterprise Resource Planning*
HTTP	*Hypertext Transfer Protocol*
IMS	*Inventory Management System*
NCD	*New Concept Development*
OMS	*Order Management System*
REST	*Representational State Transfer*
WMS	*Warehouse Management System*

1 Introdução

Este documento tem como objetivo demonstrar o trabalho realizado no âmbito da unidade curricular de Tese/Dissertação/Estágio (TMDEI) do mestrado de Engenharia Informática, com especialização em Engenharia de Software, no Instituto Superior de Engenharia do Porto.

Este projeto foi realizado em contexto empresarial, Maersk em específico, onde o principal objetivo foi dar início ao processo de migração tecnológica e arquitetural de um sistema monolítico desenvolvido em Python para um sistema assente em microsserviços desenvolvidos em .NET (C#).

1.1 Contexto de trabalho

Este capítulo pretende contextualizar o trabalho abordado neste documento através da descrição do contexto empresarial e área de atuação.

1.1.1 *HUUB*

A HUUB era uma start-up portuguesa fundada em 2015 que operava na área da logística para a indústria da moda. Esta tinha como objetivo revolucionar o "backstage" da indústria da moda, criando uma forma otimizada de gerir os processos de negócio de marcas, com foco especial no *e-commerce*, oferecendo a possibilidade destas marcas crescerem a uma escala global (*Revolutionizing the Fashion Industry - HUUB*, n.d.).

O termo "marca", utilizado de forma recorrente neste documento, é referente ao sinal utilizado para identificar e distinguir os produtos ou serviços de uma determinada empresa, sendo que, no contexto atual, representa os clientes que utilizam a plataforma. (*O Que é Uma Marca | Justiça.Gov.Pt*, 2020)

Desta forma nasceu o Spoke – uma plataforma capaz suportar todo o processo de negócio de marcas que atuam na área de *e-commerce*. Mais do que isso, o Spoke conseguia agregar todo o tipo de serviços associados à gestão de processos numa só plataforma.

Esta plataforma integrada suscitou o interesse da Maersk que, em setembro de 2021, adquiriu a totalidade da empresa. Esta aquisição resultou em múltiplas alterações: conceito, processo, tecnológico e estrutural. Apesar de todas estas mudanças, abordadas com mais detalhe e rigor em futuras secções do presente documento, a verdade é que a grande maioria dos colaboradores da HUUB continuaram na agora Maersk e, apesar da mudança ser evidente, o objetivo continua a ser o mesmo – construir uma plataforma integrada de *e-fulfillment*, agora aplicada ao contexto da Maersk.

1.1.2 *Maersk*

A empresa A.P Moller – Maersk (vulgarmente conhecida como apenas Maersk), foi fundada em 1904 na cidade de Svendborg, Dinamarca (*About A.P. Moller - Maersk | Maersk*, n.d.). Começaram por apenas possuir um navio a vapor em 2ª mão, suficiente para começar aquilo que seria um projeto para história. Em 1919 já possuíam escritórios em Nova York e a expansão continuou.

A visão desta empresa logística passa por tornar-se, não um, mas o integrador global de cadeias de suprimentos, oferecendo soluções de logísticas que se conectam e simplificam todo o processo dos seus clientes.

Os valores da Maersk são os pilares para hoje serem considerados a principal empresa de logística a nível mundial. De todos os seus valores, é importante destacar a seriedade e integridade para com os seus clientes/parceiros/funcionários.

Para além disso, a Maersk tem feito progressos incríveis naquilo que é o crescimento sustentável, através da elaboração de metas para a descarbonização dos transportes logísticos.

O setor tecnológico da Maersk têm sido um dos principais focos nos últimos anos, sendo que neste momento possui cerca de 5000 funcionários espalhados por todo o mundo. Este setor foca-se principalmente no desenvolvimento de plataformas internas e é caracterizado por utilizar metodologias ágeis de desenvolvimento de software, fazendo questão que todas as equipas espalhadas pelo mundo estão alinhadas com a visão e valores da empresa.

Neste momento, a equipa Maersk – Portugal, mais especificamente a Maersk – Porto, antiga HUUB, está responsável pelo desenvolvimento da nova plataforma global de *e-fulfillment*.

Esta equipa (Maersk - Porto) está dividida em múltiplas subequipas e, de forma a estas serem independentes, cada uma possui *backend* e *frontend developers*. Estas equipas são na sua maioria de 6 elementos: 5 desenvolveres (sendo que um é o team leader) e o *product owner*. Cada equipa é responsável por diferentes domínios, o que torna a alocação de tarefas e consequente gestão de desenvolvimento prática e fácil.

1.1.3 *Área de atuação*

A equipa em que o autor deste documento atua está responsável pela análise e migração de apenas um dos módulos presentes no monólito atual, sendo este o módulo associado à gestão de ordens, denominadas de *orders* neste documento, que são processos fulcrais naquilo que é o conceito de *EFF.*

Outras equipas irão fazer parte também deste projeto, ficando responsáveis por outros módulos. Todos estes módulos serão abordados com mais detalhe e rigor ao longo deste documento.

Os novos serviços serão obrigatoriamente desenvolvidos com base na *framework* .NET (C#) e, assim sendo, não há qualquer possibilidade de influenciar/analisar outras tecnologias para a construção destes mesmos serviços, ficando apenas disponível para análise os diferentes protolocos de comunicação a utilizar entre estes.

A infraestrutura sofre também uma alteração visto que todos os serviços são agora alojados em Microsoft Azure no lugar de AWS.

1.2 Problema

Como referido, o sucesso da plataforma criada pela HUUB levou a que, em setembro de 2021, a Maersk adquirisse a totalidade da empresa, com o principal objetivo de construir uma nova plataforma global de *e-fulfillment:* a *E-Commerce Logistics platform (ECL platform).*

Este objetivo e consequente projeto, não só obrigou a uma interação com um número muito superior de equipas, mas também a que novas equipas surgissem e a que as já existentes crescessem. Este fator originou uma reflexão cujo propósito foi tornar o desenvolvimento deste projeto o mais dinâmico possível através da otimização da arquitetura do sistema.

Visando a otimização do sistema, o *solution architect* optou por uma arquitetura baseada em microsserviços com a finalidade de, não só facilitar as divisões de responsabilidades, quer de domínio, quer entre equipas, mas também para aumentar o foco e a especialização na área de negócio em questão.

Para além disso, este projeto requer a resolução do principal problema identificado: como abordar esta migração tecnológica e arquitetural.

É importante referir que o Spoke é uma plataforma que ainda possui clientes ativos. Isto significa que, no decorrer do processo de migração, é crucial que os novos microsserviços não só tenham a capacidade de manter a antiga plataforma funcional, mas que também sejam retro compatíveis com a lógica de negócio à qual os clientes existentes estão habituados.

1.3 Objetivos e abordagem ao problema

Neste projeto está pressuposto o alcançar dos seguintes objetivos:

- **Segmentação do monólito:** O principal objetivo deste projeto passa por dar início ao processo de migração do monólito existente em diferentes microsserviços. Visando o alcançar deste objetivo, ir-se-á realizar uma análise ao sistema atual de forma a obter conhecimento acerca dos conceitos e processos existentes. Após a análise do sistema atual estar completa, o próximo passo consiste em segmentar os módulos existentes. Neste objetivo está pressuposta a finalização da análise e início do processo de segmentação do módulo de OMS (*Order Management System*) através da realização de um conjunto de requisitos, requisitos estes não definidos pelo autor deste documento.

- **Interoperabilidade:** Um dos principais objetivos e, por sinal, uma das grandes dificuldades deste projeto passa por manter a plataforma Spoke funcional até que esta esteja completamente migrada. De forma a alcançar este objetivo haverá um esforço de forma a identificar e compreender as diversas diferenças de conceitos entre as diferentes plataformas e garantir compatibilidade entre ambas. A análise e compreensão das diferenças entre os processos está fora do âmbito do projeto.

- **Estudo de diferentes abordagens:** É fundamental para o desenvolvimento deste projeto o estudo de diferentes arquiteturas. A estrutura monolítica neste momento utilizada não é uma opção pois é identificada como um problema associado à divisão de responsabilidades, quer de equipa, quer de domínio. Neste sentido, serão analisadas diferentes arquiteturas baseadas em microsserviços de forma a solucionar os problemas acima identificados. A solução escolhida será decidida com base em métodos de decisão multicritério. Também serão estudadas diferentes abordagens referentes ao processo de migração de uma aplicação monolítica para microsserviços. A abordagem escolhida deve garantir que o Spoke, plataforma que ainda possui clientes ativos, permaneça em funcionamento.

- **Resolução da dívida técnica:** Existe uma dívida técnica associada à prática do desenvolvimento de software. É possível verificar ausência de padrões, boas práticas de desenvolvimento de software e uma discrepância entre abordagens efetuadas por diferentes equipas. De forma a atingir este objetivo, serão estudadas e identificadas boas práticas de desenvolvimento de software.

- **Melhorias no processo de desenvolvimento:** Com este objetivo pretende-se aumentar a velocidade de entrega de funcionalidades através de um aumento no número de *story points* entregues. De forma a alcançá-lo, serão estudados e aplicados conceitos associados a melhorias no processo de desenvolvimento como por exemplo abordagens e modelos de qualidade, ciclo de vida de entrega de software, padrões de desenvolvimento etc.

- **Uniformização de serviços:** Tendo em conta que múltiplas equipas irão migrar diferentes serviços, há uma necessidade de permanente comunicação entre equipas de forma a uniformizá-los. Esta uniformização deverá ter em conta padrões utilizados, bibliotecas externas, etc. e será fundamental para futura manutenção dos projetos. Desta forma deverão ser estudados diferentes padrões e boas práticas que assegurem métricas de qualidade de forma a satisfazer as necessidades do projeto. O resultado deste estudo deverá ser um projeto *template* que servirá de projeto base a todos os serviços.

1.4 Restrições

Este projeto não tem apenas como objetivo migrar a plataforma existente, mas também restruturá-la e atualizá-la através da aplicação de melhorias naquilo que é o processo de desenvolvimento de software.

Neste contexto e, apesar do crescimento da equipa, a mudança de contexto de negócio leva a uma necessidade de adaptação àquilo que é o contexto da Maersk, como de seguida se irá detalhar.

1.4.1 *Restrições Tecnológicas*

Tendo em conta a mudança no contexto empresarial, seria expectável a ocorrência de alterações, que podem também ser consideradas restrições, devido à sua obrigatoriedade e resultante incapacidade de analisar possíveis alternativas:

- ***Backend*:** A Maersk impõe a obrigatoriedade na utilização de .NET (C#) para o desenvolvimento de serviços. Esta restrição afeta diretamente o desenvolvimento deste projeto no sentido em que não existe liberdade para investigar tecnologias para o desenvolvimento dos microsserviços pretendidos.
- ***Frontend*:** A Maersk impõe a obrigatoriedade de utilização de Flutter para o desenvolvimento *frontend*. Esta restrição, apesar de implicar o desenvolvimento de uma nova plataforma *web*, não afeta o desenvolvimento deste projeto e como tal, não será abordada.
- ***Cloud Services*:** A Maersk impõe a obrigatoriedade de utilização de Microsoft Azure como plataforma de serviços de nuvem. Esta restrição tem impacto direto no desenvolvimento deste projeto visto que as pipelines de automatização de processos terão de ter em conta a nova tecnologia para serviços de nuvem.

1.4.2 *Dívida técnica*

A migração tecnológica e arquitetural dá às equipas uma oportunidade de melhoria, pois torna possível a mitigação de dívida técnica previamente identificada. Atualmente, é possível

identificar padrões arquiteturais ultrapassados (ou inexistentes), ausência de boas práticas de desenvolvimento de software e indefinição naquilo que são os processos e abordagens, algo expectável em um contexto de *startup*, onde a entrega de valor é mais importante do que a engenharia de processo.

"Imaginando que tenho uma estrutura de módulos confusa na minha base de código. Preciso de adicionar uma nova funcionalidade. Se a estrutura de módulos fosse clara, levaria quatro dias para adicionar a funcionalidade, mas com esta confusão, levaria seis dias. A diferença de dois dias é dívida técnica." (Fowler, 2019)

O parágrafo anterior, citando Martin Fowler, explica quase na perfeição o porquê de a dívida técnica ser considerada uma restrição no contexto do projeto atual: a ausência de boas práticas, em conjunto com uma estrutura de código confusa e pouco clara, aliada à falta de documentação, tornam difícil o processo de análise à plataforma atual, influenciando diretamente o tempo levado para atingir esse mesmo objetivo.

1.4.3 *Dependência de serviços externos*

Tendo em conta que para construir uma plataforma de *e-fulfillment* é necessária integração com inúmeras plataformas externas, é necessário analisar e compreender de que forma estas podem influenciar negativamente o desenvolvimento do projeto.

São de realçar os seguintes tipos de plataformas:

- ***Warehouse Management System* (WMS)**: No contexto da gestão de armazém, é necessário compreender que existem dependências naquilo que são os sistemas dos múltiplos armazéns externos com os quais o Spoke integrava.
- ***Enterprise Resource Planning* (ERP)**: Em contexto de gestão de pessoas, podem existir dependências naquilo que são os diferentes ERPs utilizados pelas diferentes marcas.
- ***E-commerce platforms***: Visto que o *e-fulfillment* pretende, se forma resumida, automatizar, simplificar e melhorar processos relacionados com *e-commerce*, é necessária a integração com plataformas de vendas online (Shopify, Woocomerce etc.).

1.5 Código de Ética e Conduta

Esta secção tem como objetivo definir o código de ética e conduta de forma a assegurar que todos os envolvidos neste projeto mantenham elevados padrões de ética e conduta profissional.

O Código de ética e conduta estabelece os princípios éticos e regras que devem ser cumpridas no decorrer todo o projeto de forma a proteger os interesses de todos os envolvidos.

Desta forma, são tidos em conta os seguintes princípios:

- **Honestidade e Integridade**: Todos os envolvidos devem agir com honestidade e integridade em todos os aspetos do projeto.
- **Confidencialidade**: Todos os envolvidos devem manter a confidencialidade de qualquer informação confidencial que seja partilhada com eles no decurso do projeto.
- **Respeito pelos outros**: Todos os envolvidos devem ser tratados com respeito, evitando discriminação, assédio, ou qualquer outra conduta que seja desrespeitosa ou ofensiva.
- **Responsabilidade**: Todos os envolvidos são responsáveis pelas suas ações e decisões, e devem tomar as medidas apropriadas para assegurar que o projeto seja concluído com êxito.
- **Cumprimento das leis e regulamentos**: Todos os envolvidos devem cumprir as leis e regulamentos aplicados no momento, incluindo as leis de propriedade intelectual e as leis de proteção de dados.
- **Profissionalismo**: Todos os envolvidos devem manter padrões elevados de profissionalismo.

De forma a garantir todos estes pontos existe uma constante comunicação entre as partes envolvidas neste documento, garantindo assim que todos os princípios e regras são cumpridos conforme planeado.

1.6 Estrutura do documento

Visando documentar todo o processo de desenvolvimento do projeto anteriormente referido, o presente documento encontra-se dividido nos seguintes capítulos:

1. **Introdução**: É abordado de forma breve o contexto de trabalho, o problema a resolver, os objetivos e a abordagem ao problema.
   São também apresentadas as restrições existentes no projeto.
2. **Estado da arte:** É descrita a área de atuação do projeto em questão, seguida de uma abordagem às tecnologias utilizadas, bem como uma análise científica a artigos relacionados com o tópico em questão.
3. **Análise de valor:** É avaliado o valor do projeto através da utilização de variadas ferramentas de análise e avaliação.
4. **Análise e conceção:** É abordada a atualidade do sistema, seguidas de propostas de arquiteturas resultantes da migração do monólito descrito através do recurso a diagramas UML. Depois, as abordagens são comparadas através da aplicação de um método multicritério (AHP) e uma será escolhida para desenvolvimento.
5. **Desenvolvimento:** São apresentadas as soluções desenvolvidas, tendo por base o processo de análise e conceção.
6. **Experimentação e Avaliação:** São abordadas as hipóteses de avaliação e identificados os indicadores, fontes de informação e metodologias adotadas.
7. **Conclusões:** São abordados e analisados o nível de cumprimentos dos objetivos propostos assim como possíveis considerações de trabalho futuro.

2 Estado da Arte

O objetivo deste capítulo é dar uma visão completa dos conceitos associados ao tema deste projeto, analisando-os e, quando possível, efetuar comparações com possíveis alternativas. Ao investigar os conceitos básicos e teorias que o sustentam, pretende-se construir uma base sólida para fundamentar a tomada de decisão.

Desta forma é realizada uma pesquisa dos conceitos, modelos e padrões mais utilizados, bem como uma análise de suas vantagens e desvantagens. Este capítulo também apresenta uma visão geral da área de atuação na qual se enquadra este projeto.

2.1 Métodos de pesquisa

Como foi referenciado, o objetivo do estado da arte passa por fundamentar teoricamente os conceitos associados ao tema deste projeto. Para tal, são utilizados, de forma conjunta, diferentes tipos de métodos de pesquisa, abordados nos seguintes parágrafos.

A primeira etapa, consideravelmente menos autónoma, passou por pedir aconselhamento, quer ao orientador deste projeto, quer a colegas de trabalho, sobre possíveis documentos científicos, livros e autores que pudessem contribuir de forma positiva para o desenvolvimento deste documento. Esta etapa foi fundamental para fundamentar muitos conceitos associados a paradigmas arquiteturais, padrões de migração e até mesmo boas práticas de desenvolvimento de software.

Em seguida, a fase de exploração, onde de forma autónoma e, com recurso a ferramentas como *google scholar*, *IEEE explore* etc., efetua-se pesquisas com recurso às palavras-chave

identificadas para o projeto em questão: e-commerce, e-fulfillment, microsserviços, monolítico, arquitetura, software, .NET, Python.

Por fim, o método menos utilizado, porém também importante: recurso a *blogs* e *sites* de informação pontual de forma a encontrar informação relacionada a, por exemplo, estatísticas.

2.2 Área de atuação

Este capítulo pretende contextualizar a área de atuação do projeto atual, de forma a obter-se uma melhor compreensão de conceitos fundamentais utilizados no decorrer de todo o documento.

2.2.1 *E-commerce*

Comércio eletrónico, também conhecido como *e-commerce*, é o termo utilizado para descrever a compra e venda de bens e serviços pela internet.

Desde a sua aparição, mudou completamente a forma de como a população faz compras e realiza negócios, proporcionando conforto, acessibilidade, e uma maior variedade de possibilidades. O comércio eletrónico expandiu-se exponencialmente nos últimos anos como resultado de melhorias tecnológicas, aumento da utilização da Internet, e mudanças nos padrões de consumo (*Ecommerce Defined: Types, History, and Examples*, 2022).

Prevê-se que o negócio do comércio eletrónico mundial continue o seu crescimento em popularidade. A pandemia originada pela COVID-19 aumentou ainda mais a tendência de compra pela Internet, uma vez que esta permite aos clientes adquirirem produtos sem a necessidade de se deslocarem ao local de venda.

O *e-commerce* tem criado oportunidades, assim como novos problemas para empresas de todas as dimensões. Esses desafios enfrentados pelas empresas vão desde a concorrência à escala mundial até às regulamentações complexas e questões de segurança. Além disso, todas as empresas viram-se obrigadas a investir em *websites* e aplicações *mobile* com o objetivo de acompanharem o crescimento do comércio online**.**

Não obstante destes obstáculos, este setor continua a expandir-se e a mudar devido à inovação e às crescentes exigências dos consumidores. Surge agora a oportunidade de integração com a realidade virtual e aumentada e a inteligência artificial pronta a revolucionar a forma como os consumidores fazem compras e realizam negócios online.

2.2.2 *E-fulfillment*

O avanço da tecnologia e a utilização generalizada da Internet levaram a um aumento significativo da popularidade do *e-commerce*, como já foi mencionado previamente neste

documento. A capacidade dos consumidores de comprar bens e serviços a partir do conforto das suas próprias casas tornou as compras online uma opção conveniente e atraente para muitas pessoas. No entanto, com o aumento do *e-commerce*, surge também o desafio de satisfazer as necessidades dos clientes de uma forma eficiente e rentável.

É precisamente neste ponto que entra o *e-fulfillment* (*E-Commerce Logistics & Order Fulfilment Services | Maersk*, n.d.). Este é uma solução integrada para os desafios lançados pelo crescimento do *e-commerce* e pretende suportar o processo de receção, armazenamento, seleção, embalamento e distribuição de produtos (*5 Biggest Order Fulfillment Challenges That Ecommerce Companies Face - Amazon Multi-Channel Fulfillment (MCF)*, 2022).

Desta forma consegue-se aliviar, ou até mesmo resolver vários problemas do mundo real enfrentados pelas empresas que possuem *e-commerce*, tais como:

- **Ineficiência no armazenamento**: Permite que as marcas contornem os custos e limitações associados ao armazenamento, consequentemente reduzindo as despesas operacionais e aumentando a eficiência.
- **Pouca procura**: Permite que as marcas entreguem produtos diretamente aos seus clientes, proporcionando-lhes uma alternativa conveniente.
- **Pouco alcance**: Permite às empresas chegar aos clientes em qualquer parte do mundo, aumentando assim o seu número de possíveis clientes.
- **Gestão de inventário**: Permite às marcas gerir o seu inventário de forma mais eficiente, oferecendo uma visão completa a nível de *stock*, ordens, vendas, devoluções, entre outros.
- **Capacidade de resposta**: Oferece às marcas uma maior capacidade de gestão de elevados volumes de pedidos.
- **Experiência do cliente**: Ajuda as empresas a proporcionar aos seus clientes uma boa experiência de compra, o que pode originar um maior número de clientes habituais e a uma melhor reputação.

2.3 Análise Arquitetural

Com o objetivo de fundamentar as decisões arquiteturais, é fundamental contextualizar não só o tipo de arquitetura utilizada pelo sistema atual, como também o tipo de arquitetura para a qual se pretende transacionar.

A arquitetura de software refere-se à estrutura de um sistema e abrange os componentes, módulos e interfaces que o constituem, bem como as relações entre os mesmos. É a estrutura na qual um software se define em termos técnicos e operacionais. É, também, o principal responsável pela eficiência, capacidade de gestão, escalabilidade, confiabilidade, modificabilidade, entre outros aspetos, de uma aplicação. Por isso mesmo é que a escolha da arquitetura a utilizar é tão importante na fase primária de desenvolvimento de software (*What Is Software ArchItecture: A Guide | Built In*, 2022).

As seguintes secções têm como objetivo contextualizar o tipo de arquitetura utilizada no sistema atual (monolítica) e o tipo de arquitetura para a qual se pretende transacionar (orientada a microsserviços).

2.3.1 *Arquitetura monolítica*

Uma aplicação monolítica é um sistema de software projetado para ser independente e autossustentável. É um sistema único e autónomo construído como uma unidade única, em vez de ser composto de componentes modulares menores (Tapia et al., 2020).

A seguinte imagem pretende representar um exemplo daquilo que poderia ser uma arquitetura monolítica de um sistema de *e-commerce*:

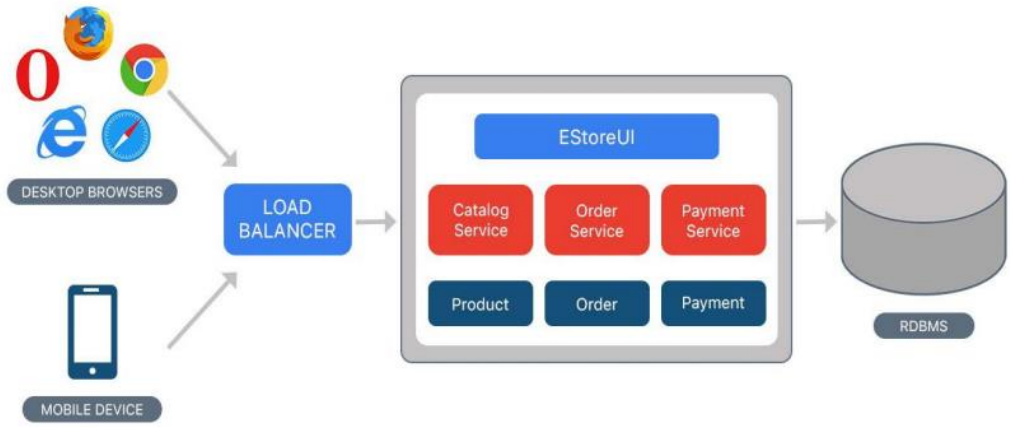


Figura 1 - Arquitetura monolítica (*Microservices vs Monolithic Architecture - What Should You Choose? | XME Advice*, 2022)

Esta arquitetura apresenta um conjunto de vantagens facilmente identificáveis. Visto ser o modelo mais utilizado durante anos, não há falta de documentação pela qual se pode fundamentar decisões. Não só torna mais fácil o desenvolvimento, como também a sua implementação em um ambiente funcional. A testabilidade também é elevada devido à inexistência de dependências externas.

Por sua vez, as desvantagens podem ser inumeradas da seguinte forma:

- Aplicações tornam-se grandes e difíceis de gerir.
- É necessário voltar a implantar a aplicação inteira, por mais pequena que seja a alteração, visto que toda a aplicação está assente em único artefacto.
- Com o aumento do tamanho e aglomeração de testes, o processo de CI/CD torna-se incrementalmente mais lento.
- Não existe a possibilidade de segregar tráfego visto que é necessário implantar a aplicação inteira.

- O crescimento da aplicação leva a um enorme acoplamento entre diferentes serviços, dificultando não só o desenvolvimento de novas funcionalidades como também o processo de alterações em funcionalidades já existentes.
- Torna a inclusão de novos colaboradores mais difícil no sentido em que a solução é muito abrangente e pouco focada.
- Complica a gestão da solução pois a submissão de código é toda feita no mesmo repositório. Pode levar a descoordenação e tornar todo o processo de CI/CD mais confuso.
- Impõe restrições tecnológicas visto que se trata de uma única solução.
- Torna as aplicações menos confiáveis visto que um único erro pode fazer com que a aplicação fique indisponível.

Tendo por base as características citadas, existem ainda vários tipos de sistemas monolíticos, sendo os principais (Newman, 2019):

- ***The Single Process Monolith*:** é provavelmente o mais comum tipo de sistema monolítico, onde toda a base de código encontra-se junta e a aplicação é toda implantada num único processo.
- ***Modular monolith*:** é uma alternativa ao *single process monolith*, em que a base do código deixa de estar junta e passa a estar separada por módulos. Apesar desta divisão modular, o processo da implantação continua a ser conjunto.
- ***The Distributed Monolith:*** é uma aplicação que é implantada como um microsserviço, mas que é construída como um monólito. Neste tipo de arquitetura, tem-se o peso e a inflexibilidade dos monólitos, a complexidade dos microsserviços, e poucos dos benefícios de qualquer das arquiteturas. Para além disso, tem-se ainda implantações lentas e fraca escalabilidade.

2.3.2 *Arquitetura orientada a microsserviços*

A arquitetura orientada a microsserviços é um padrão de *design* de *software* no qual um sistema é construído como um agregado de serviços menores. Estes serviços são projetados para serem unidades modulares e independentes que podem ser desenvolvidos, implantados e dimensionados independentemente uns dos outros (Fowler, 2014d).

Normalmente, este tipo de arquitetura pretende que sejam disponibilizados *endpoints* a partir dos quais outros serviços podem comunicar (Fowler, 2014d).

Um dos principais benefícios de arquiteturas baseadas em microsserviços é que estas permitem uma maior flexibilidade e agilidade no processo de desenvolvimento visto que cada serviço é autónomo e pode ser desenvolvido e implantado de forma independente, sendo que as equipas podem trabalhar em diferentes partes do sistema simultaneamente, sem a necessidade de coordenar esforços ou esperar que outras partes do sistema sejam concluídas. Este facto torna

o processo de desenvolvimento célere e facilita a inclusão de novos recursos ou alterações no sistema.

Outra vantagem de arquiteturas orientadas a microsserviços é que estas permitem uma melhor e mais fácil manutenção do sistema. Como cada serviço é independente, pode ser ampliado ou reduzido conforme necessário, sem afetar os restantes componentes do sistema. Este facto ajuda a tornar o sistema capaz de lidar com aumento de tráfego e pode facilitar a identificação e correção de problemas aquando da sua ocorrência (Newman, 2019).

Em contrapartida, é importante compreender que uma arquitetura orientada a microsserviços não é a solução ideal para todos os sistemas. Estes tipos de sistema são, por norma, mais complicados de desenhar e implementar e podem, na grande maioria das ocasiões, exigir mais recursos e infraestrutura para gerir e manter o sistema. Além disso, a comunicação e a coordenação entre os diferentes serviços podem ser complexas, tornando difícil garantir que o sistema está a funcionar da maneira esperada.

Sendo assim, decidir se a arquitetura de microsserviços é ou não a escolha certa para um determinado sistema, dependerá de vários fatores, incluindo o tamanho e a complexidade do sistema, os recursos disponíveis e as necessidades dos próprios utilizadores. As arquiteturas orientadas a microsserviços são adequadas para sistemas maiores e mais complexos que exigem um elevado grau de flexibilidade, escalabilidade e manutenibilidade.

Em suma, arquiteturas baseadas em microsserviços são:

- Desenhadas e implementadas com alto nível de granularidade, em que cada serviço pretende executar tarefas específicas. Isto torna o sistema fácil de gerir e manter pois cada um destes têm um propósito específico.
- Implementadas utilizando tecnologias agnósticas como por exemplo REST API ou protocolos baseados na produção/consumo de mensagens como AMQP ou Kafka. Isto torna a comunicação entre os diferentes serviços flexível e interoperável.

Desta forma, a seguinte lista pretende enumerar múltiplos padrões associados a arquiteturas orientadas a microsserviços:

- ***Database per Service*:** Este padrão é normalmente utilizado em sistemas orientados a microsserviços e consiste na existência de uma base de dados individual para cada um dos serviços (Richardson, 2018c). Na prática, isto significa que cada serviço é responsável pela gestão dos seus próprios dados e que não há uma base de dados partilhada ou centralizada para todo o sistema. Existem várias vantagens em utilizar *database per service*, sendo que uma das principais é permitir que cada serviço tenha total controlo sobre seus próprios dados, o que pode facilitar a gestão do sistema. Outra vantagem deste padrão é que este pode ajudar a melhorar o isolamento e a modularidade do sistema pois as alterações nos dados de um serviço não afetarão necessariamente os dados de outros serviços, facto este que pode ajudar a reduzir o risco de consequências não intencionais ao fazer alterações no sistema. Em

contrapartida, é importante ter em conta que esta abordagem pode apresentar alguns desafios, como por exemplo a realização de consultas ou transações que abrangem vários serviços, uma vez que os dados estão distribuídos em múltiplas bases de dados.

- ***Event sourcing:*** É um padrão de arquitetura de software que envolve o armazenamento de mensagens em uma "fila" de eventos. Este é um padrão frequentemente utilizado em arquiteturas orientadas a microsserviços pois apresenta inúmeros benefícios como, por exemplo, permitir um registo completo das alterações que ocorrem no sistema, o que facilita *troubleshooting* e resolução de bugs (Richardson, 2018d). Há algumas considerações importantes a terem em conta na utilização deste padrão visto que a sua implementação é consideravelmente mais complexa do que utilizar o protocolo REST e exige uma maior e mais complexa infraestrutura visto que a solução de *event sourcing* também necessita ser implementada, implantada e mantida.
- ***CQRS (Command Query Responsibility Segregation):*** É um padrão arquitetural que separa as operações de consulta das operações que atualizam os dados (comandos), dividindo o sistema em duas secções distintas (Richardson, 2018b). A secção dos comandos é responsável pela modificação das entidades pelas quais o sistema está responsável, enquanto a secção de consulta é responsável pelo fornecimento de informação sobre essas mesmas entidades. Este padrão normalmente utilizado em conjunto com o padrão "*Event Sourcing"* e tem como principal objetivo aumentar a escalabilidade do sistema.
- ***API gateway pattern***: Este padrão envolve o uso de uma API *gateway* que redireciona os pedidos para os respetivos microsserviços. Funciona como um único ponto de entrada para todos os pedidos, o que ajuda a centralizar algumas funcionalidades como autenticação, *rate limiting* e *caching*.
- ***Circuit breaker pattern***: Padrão utilizado para prevenir falhas em um determinado serviço, definindo um limite máximo de tentativas falhados para um determinado período. Quando este limite é atingido, o *circuit breaker* é ativado e todas as chamadas àquele determinado serviço falham (Richardson, 2018a).
- ***Load balacing pattern:*** Este padrão é utilizado para distribuir os pedidos a um determinado serviço por múltiplas instâncias desse mesmo serviço, de forma a melhorar a performance e disponibilidade do mesmo.
- ***Service discovery pattern*****:** Envolve o uso de um registo de serviços que guarda a localização e estado atual dos mesmos de forma a ajudar na motorização e localização.
- ***Health check pattern:*** Este padrão consiste em expor um *endpoint* que devolve o estado atual do serviço.

2.4 Segmentação de um sistema monolítico

A segmentação de um sistema monolítico refere-se ao processo de decomposição de um sistema grande em componentes mais pequenos e mais modulares. Isto pode ser feito por uma variedade de razões, incluindo a melhoria da escalabilidade, manutenção e flexibilidade do sistema.

Domain-driven design (DDD) é o ponto de partida para embarcar na jornada de segmentação de um sistema monolítico. Esta transformação de monolítico para microsserviços não é apenas uma melhoria naquilo que é a arquitetura do sistema, mas também uma oportunidade de reorganizar o modelo de forma a melhor satisfazer as necessidades do sistema.

DDD é o mecanismo que nos permite compreender os limites de um serviço, isto é, que parte do sistema deve ou não deve um determinado serviço cobrir (Fowler, 2014c).

Consiste em um conjunto de princípios e padrões que permitem aos desenvolvedores criar sistemas orientados a objetos. Aplicado corretamente pode originar modelos de domínio que encapsulam regras de negócio complexas.

Para tal, é necessário ter em conta alguns conceitos impostos por esta abordagem de *design* de software, entre os quais:

- ***The need for a Common Language (the ubiquitous language)*:** Um projeto pode encontrar variados problemas quando os membros de equipa não partilham uma linguagem comum naquilo que é a discussão do domínio. Este conceito é referente à linguagem que especialistas de domínio, desenvolvedores e partes interessadas utilizam quando falam do domínio em questão.
- ***Bounded contexts:*** Estes são definidos pela criação dos limites de modelo e a relação entre os vários elementos deste. Os *bounded contexts* pressupõe a definição de agregados de diferentes entidades que se relacionam.
- ***Entities:*** Representam objetos de domínio e tentam assemelhar-se o mais possível à vida real.
- ***Value objects:*** Objetos que são utilizados como atributos dos objetos de domínio, não existindo por si próprio nem possuindo uma identidade própria.

O DDD auxilia naquilo que é tarefa de definir os diferentes subdomínios do sistema, isto é, o domínio de um microsserviço.

Agora que DDD está definido, podemos utilizar os seus conceitos para o primeiro passo da segmentação de um monólito, análise de domínio, onde se pretende obter uma abstração daquilo que e é modelo de negócio do sistema que se pretende desenvolver. O *output* deste primeiro passo deverá ser um modelo no qual se consegue compreender os subdomínios e como estes se integram entre si e, caso necessário, também com componentes externos.

Assim que a análise do domínio está completa, é necessário identificar os *bounded contexts*, que representam de uma forma mais simples aquilo que são os limites do modelo de domínio (Fowler, 2014a).

A partir dos passos supracitados é possível identificar e definir os diferentes microsserviços do novo sistema. Tipicamente, um *bounded countext* origina um microsserviço, porém existe a possibilidade de um *bounded context* originar mais que um microsserviço, no caso de este possuir agregados que por si só possam ser candidatos a tal.

2.5 Padrões de migração

Antes de iniciar o processo de migração e, com o objetivo de fundamentar as decisões tomadas, é essencial analisar e compreender os principais padrões de migração, de forma a tomar conhecimento das principais vantagens e desvantagens dos mesmos e, escolher qual deles se melhor adequa à situação em questão.

2.5.1 *Strangler pattern*

O *Strangler Pattern* é frequentemente usado com o intuito de reduzir o risco (Fowler, 2014e).

Através da aplicação deste padrão é possível migrar lógica de um monólito para um módulo independente e, neste padrão é normal que, por algum tempo, a implementação antiga e a nova existam em simultâneo, porém, eventualmente, a solução antiga deixará de fazer sentido pois a nova irá substituí-la por completo.

Novamente utilizando um exemplo de um sistema monolítico de *e-commerce* como exemplo:

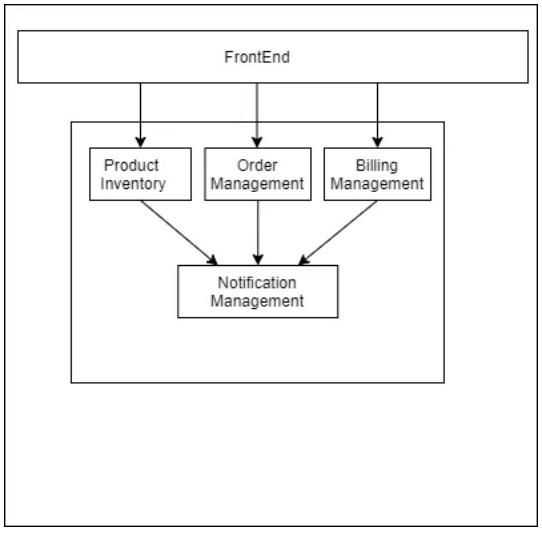


Figura 2 - Exemplo de Arquitetura monolítica (Kapoor, 2021)

Tendo como exemplo a Figura 2, e tendo o módulo "*Billing Management*" como exemplo, seguem-se os seguintes passos:

- Implementar uma API *gateway* que consiga redirecionar os pedidos feitos para o novo serviço. Nesta etapa inicial, a *gateway* apenas servirá como um *passthrough layer* e irá redirecionar os pedidos para o monólito como é possível visualizar na seguinte imagem:

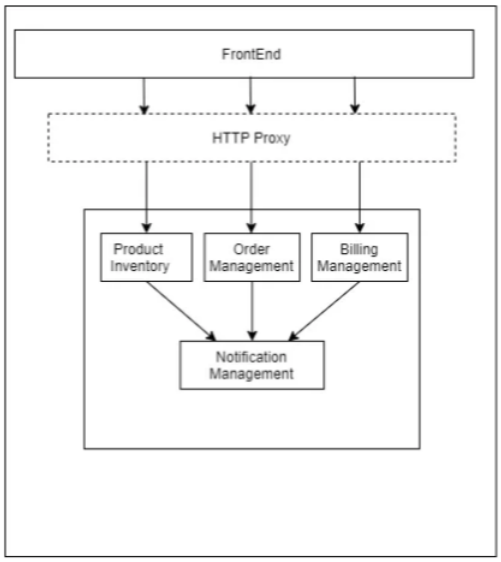


Figura 3 - Strangler Pattern API gateway (Kapoor, 2021)

- Desenvolver o novo microsserviço, utilizando a tecnologia adequada e aplicando boas práticas de desenvolvimento de software de forma a eliminar a dívida técnica existente.

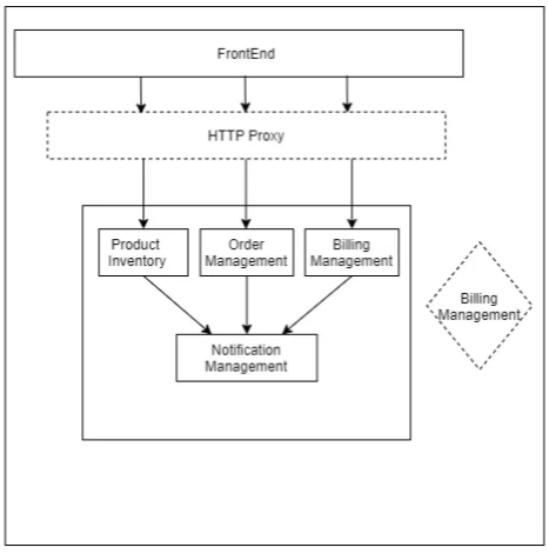


Figura 4 - Strangler Pattern desenvolvimento do novo serviço (Kapoor, 2021)

- Alterar a API Gateway de forma a redirecionar os pedidos para o novo serviço. Esta etapa é fulcral e, caso ocorra algum problema, é possível voltar a alterar a *gateway* de forma a voltar a redirecionar o tráfego para o monólito:

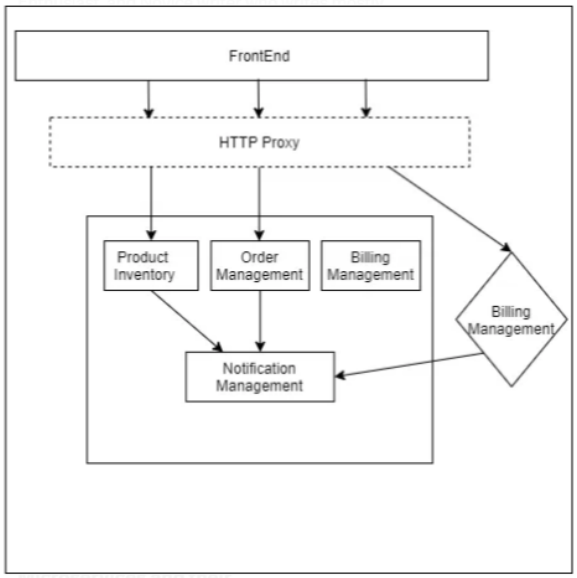


Figura 5 - Strangler Pattern redirecionamento de tráfego (Kapoor, 2021)

- Assim que o comportamento do novo serviço esteja conforme esperado, o módulo "*Billing management*" pode ser removido do monólito, completando-se assim o processo de migração de um dos módulos existentes.

2.5.2 *Branching By Abstraction*

Este padrão permite que se mova um módulo interno da aplicação monolítica para um microsserviço autónomo. O módulo não lida diretamente com o tráfego e é utilizado internamente por outros módulos.

Este padrão também permite que, quer a implementação da funcionalidade existente, quer o novo serviço, coexistam durante algum tempo. Um exemplo poderia ser o Módulo de Notificações da aplicação *e-commerce* acima exemplificada.

Para implementar este padrão seguem-se os seguintes passos:

- Criar um ponto de abstração para o módulo que irá ser migrado para microsserviço e alterar a implementação existente de forma que se passe a usar a abstração e não o modulo diretamente (Fowler, 2014b), como é possível visualizar na seguinte imagem:

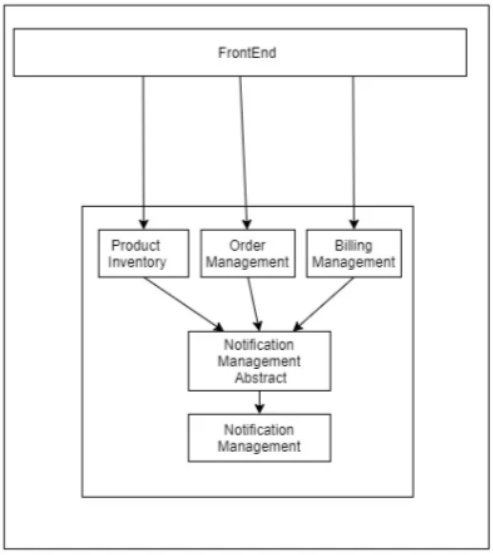


Figura 6 - Branching by abstraction (Kapoor, 2021)

- Desenvolver um novo microsserviço que corresponda ao módulo que se pretenda migrar:

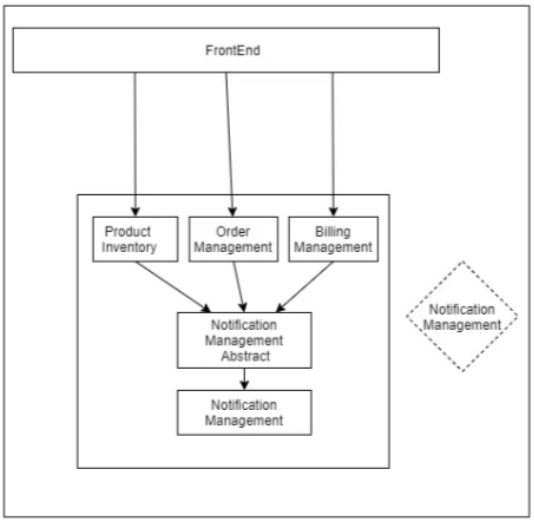


Figura 7 - Branching by abstraction new service (Kapoor, 2021)

- Assim que o novo serviço tenha o comportamento esperado, pode-se concluir o processo de migração, redirecionando o tráfego para o novo serviço e eliminando o módulo previamente existente:

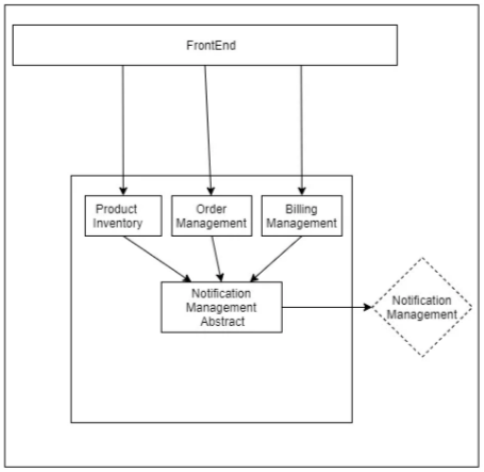


Figura 8 - Branching by abstraction redirecionamento de tráfego e remoção do módulo (Kapoor, 2021)

2.5.3 *Parallel run*

Migração em paralelo, ou *parallel run,* é um padrão também utilizado para migrar sistemas monolíticos para sistemas orientados a microsserviços, porém com uma grande diferença em relação aos restantes: ambos os sistemas estão em funcionamento, em paralelo, durante um determinado período, de forma assegurar o funcionamento correto do novo sistema antes que a transição seja efetuada.

Este padrão, ao colocar os dois sistemas em funcionamento em paralelo, permite comparar os resultados obtidos pelo novo microsserviço aos resultados obtidos pelo monólito.

Desta forma, consegue-se minimizar os riscos durante o processo de migração, permitindo uma migração gradual e segura dos serviços desejados.

Para além da segurança proporcionada por este processo, é importante ter em conta que, os sistemas, ao coexistirem, exigem recursos adicionais, aumentando o custo e complexidade associada à gestão dos mesmos.

Inicialmente, a fonte da verdade será sempre o monólito, até que se consiga perceber que o novo serviço está de facto pronto, tal como pretende ilustrar a seguinte figura:

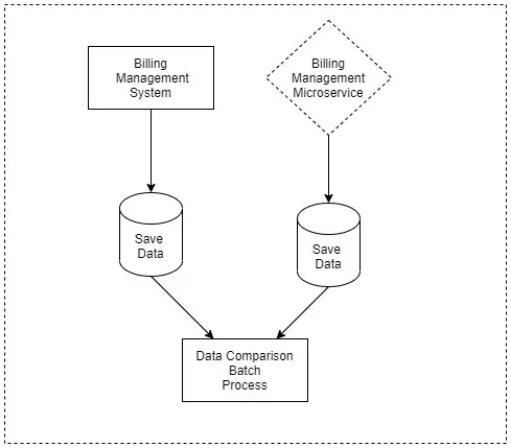


Figura 9 - Parallel run

2.6 Análise tecnológica

Com base nas restrições tecnológicas referidas no primeiro capítulo deste documento e, tendo em conta que a utilização de .NET (C#) é de carácter obrigatório, não é de carácter relevante analisar possíveis alternativas a esta tecnologia.

No entanto, é de caracter relevante analisar a utilização atual de protocolos de comunicação. Sendo assim o próximo capítulo descreve os protocolos de comunicação utilizados neste momento, sendo destes HTTP/REST e Apache Kafka.

2.6.1 *Protocolos de comunicação*

Microsserviços são um estilo arquitetural que permite construir sistemas modulares que comunicam entre si de forma a compor um sistema completo.

Visto que estes serviços independentes entre si, está pressuposto que estes consigam comunicar de diferentes formas, sendo que as adotadas para este projeto são:

2.6.1.1 HTTP and REST

Hypertext Transfer Protocol é um protocolo utilizado para a transmissão de *hypertext documents*, como por exemplo, HTML.

Foi concebido para a comunicação entre *browsers* e *web servers*, contudo, ao dia de hoje, é utilizado em variados contextos.

HTTP segue um modelo clássico de cliente-servidor, onde um cliente abre uma conexão, realiza um pedido e espera uma resposta. Este protocolo é considerado um protocolo sem estado no sentido em que nem o cliente nem o servidor guarda qualquer tipo de informação em relação aos pedidos efetuados e respostas recebidas (*HTTP | MDN*, 2022).

É o protocolo de comunicação mais utilizado pelos desenvolvedores de software devido à simplicidade de implementação e maturidade no mercado.

REST *(Representational State Transfer*) é um estilo arquitetural utilizado para a construção de serviços e pode ser descrito como um conjunto de restrições e padrões que tendem a ditar a forma de como esses mesmos serviços operam. Estes serviços são construídos com base no protocolo HTTP e utilizam uma lista definida de verbos (*get, put, post, delete etc.*) para realizar operações CRUD (*Create*, *Read*, *Update*, *Delete*).

2.6.1.2 Apache Kafka

Um *message broker* é um componente de software que funciona como mediador entre aplicações que necessitam de enviar e receber mensagens (neste contexto, os microsserviços). Estes fornecem um ponto de entrada para aplicações que enviam e recebem mensagens de forma assíncrona, o que pode ser útil em situações em que a comunicação em tempo real não é necessária ou em que as aplicações precisam de ser dissociadas umas das outras.

Eis algumas das principais características e benefícios dos *message brokers*:

- **Escalabilidade**: podem lidar com um grande volume de mensagens e podem ser escalados para suportar ambientes de alta concorrência.
- **Confiança**: podem armazenar e reencaminhar mensagens mesmo que uma ou mais aplicações não estejam disponíveis, assegurando que as mensagens não se percam ou sejam perdidas.
- **Flexibilidade**: podem suportar diferentes protocolos de mensagens e formatos de dados, permitindo às aplicações comunicar independentemente da sua tecnologia ou arquitetura subjacentes.
- **Integração**: podem facilitar a integração de diferentes aplicações e sistemas, fornecendo uma plataforma de mensagens comum.

Apache Kafka baseia-se num *cluster* composto por vários *brokers*, que são responsáveis pela receção, armazenamento e distribuição de mensagens. Estas mensagens são produzidas com base em um modelo de publicação-subscrição, onde os produtores publicam mensagens para tópicos previamente definidos. De forma a estas mensagens serem processadas, vários sistemas/serviços podem subscrever ao tópico definido de forma a consumir as mensagens nele publicadas. Os tópicos Kafka são particionados, permitindo desta forma que inúmeras mensagens sejam processadas simultaneamente.

Apesar de ser um *message broker* extremamente completo, oferecendo um vasto leque de funcionalidades capazes de satisfazer a grande maioria dos casos de uso, a sua elevada complexidade é um fator a ter em consideração.

2.7 Métricas de qualidade

Este capítulo pretende identificar e analisar as principais métricas de qualidade, que podem ser descritas como parâmetros e medidas que têm como objetivo garantir melhorias na qualidade do desenvolvimento e consequente melhoria no produto final e tempo de entrega.

2.7.1 *Abordagens de Qualidade*

São necessárias duas abordagens para que uma equipa de desenvolvimento de software tenha qualidade no desenvolvimento do mesmo: **QA** (*Quality Assurance*) e **QC** (*Quality Control*) que, se usadas em simultâneo, garantem que as funcionalidades desenvolvidas e entregues sejam de qualidade, com melhor aceitabilidade e no menor tempo possível.

QA é um processo de gestão, em que os desenvolvedores com mais experiência devem ter visibilidade apropriada sobre o processo de desenvolvimento de software, garantindo os padrões de qualidade do mesmo. **QC** têm como objetivo o teste de funcionalidades de modo a encontrar e corrigir possíveis *bugs* (Hamilton, 2023).

A avaliação de um produto de software é um dos processos do tempo de ciclo do mesmo e contribui para a melhoria na qualidade do produto.

2.7.2 *Tempo de Ciclo*

Tempo de ciclo é um termo utilizado para definir um conjunto de etapas que ocorrem entre o desenvolvimento de uma funcionalidade até esta ser concluída. O termo “concluído” pode variar dependendo do contexto e da “definição de feito” considerada.

Atualmente, todos os dados referentes ao processo de desenvolvimento são facilmente consultados visto que existem inúmeras ferramentas de auxílio à gestão de projetos, como por exemplo a plataforma JIRA.

Visando a diminuição do tempo associado aos ciclos de vida, faz sentido estudar diferentes possibilidades que visam esse mesmo objetivo. A automatização de processos é fundamental para o processo de desenvolvimento de software e ajuda a diminuir o tempo de ciclo da entrega de software. Entre todos os processos de desenvolvimento de software que podem ser automatizados, destacam-se os seguintes:

2.7.2.1 CI/CD Pipeline

É um dos componentes fundamentais ao desenvolvimento de software automatizado. CI/CD é uma estratégia de desenvolvimento de software que aumenta a velocidade de desenvolvimento ao mesmo tempo que assegura que a qualidade do código implantado não é comprometida.

Continuos Integration (CI) automatiza o processo de *building* e *testing* do software desenvolvido. É um ponto fulcral do desenvolvimento de software pois múltiplos desenvolvedores contribuem com código em um repositório partilhado. Apesar de parecer simples, introduz complexidade ao nível do controlo de versões. É esta complexidade que o processo de CI tende a simplificar, visto que garante que o código é compilado e testado antes de ser *merged,* podendo depois avançar para a fase de *deployment*. É um processo fundamental que garante não só a continuidade de funcionalidades já existentes, como a deteção de bugs.

Continuos Deployment (CD) permite o *deployment* automático do software após o processo de CI. É onde se pode orquestrar *deployments* para produção ou outros ambientes.

2.7.2.2 Análise Estática

O desenvolvimento de software está, normalmente, associado a um vasto conjunto de regras e padrões que devem ser respeitados. Apesar de não ser uma obrigatoriedade, trata-se de convenções que facilitam, por exemplo, a leitura e manutenção do projeto.

Visto que estas regras são conhecidas, é um excelente processo que pode ser automatizado, por exemplo, aquando da realização de um *pull request*.

Existem múltiplas ferramentas de *static analysis*, sendo que a que está neste momento em uso é o SonarQube (Self-managed | SonarQube | Sonar, 2023). É uma ferramenta utilizada a nível global que permite analisar a qualidade de inúmeras linguagens de programação. Está desenvolvida para ser integrada com os sistemas de CI/CD e é realizada aquando da realização de um *pull request*. Caso este não respeite os padrões definidos, torna-se impossível de realizar *merge* até que os problemas estejam corrigidos.

2.7.2.3 Templates

A arquitetura de microsserviços pressupõe a existência de inúmeros projetos de software. Este processo, feito manualmente, é um processo demorado e ineficiente para os níveis que se pretendem atingir.

Desta forma, a criação de *templates* de projetos pode facilitar o processo de criação de um novo serviço. Esta estratégia permite não só uma melhoria no tempo de ciclo da entrega de um novo serviço, mas também a uniformização destes, visto que todos os serviços terão por base o mesmo projeto.

A criação de um *template* possui complexidade associada à escolha de um padrão de desenvolvimento que será utilizado por todos os microsserviços desenvolvidos pelas diferentes equipas. Os fundamentos teóricos para a escolha do padrão serão abordados na secção "Padrões de desenvolvimento."

2.7.2.4 Monitorização de erros

Erros são inevitáveis quando se fala de desenvolvimento de software. Este pressuposto torna o uso de soluções de monitorização essenciais para que se consiga ter visibilidade de erros e bugs da aplicação, principalmente após esta estar *deployed*.

Existem múltiplas ferramentas de suporte à monitorização de erros, sendo que a que está neste mento em uso é denominada de Sentry (*Home | Product Blog • Sentry*, n.d.). Esta ferramenta, para além de dar visibilidade aos erros, também os agrupa por tipo, ambiente, número de ocorrências e *release*. Como alternativa foi também considerada a utilização de uma plataforma de observabilidade denominada Grafana (*Grafana: The Open Observability Platform | Grafana Labs*, n.d.), sendo apenas descartada por questões relativas a limitações de tempo.

2.7.3 *Modelo de qualidade*

O modelo de qualidade categoriza a qualidade de software em 6 diferentes características, características estas que podem ainda ser subdivididas, tal como pretende demonstrar a seguinte imagem:

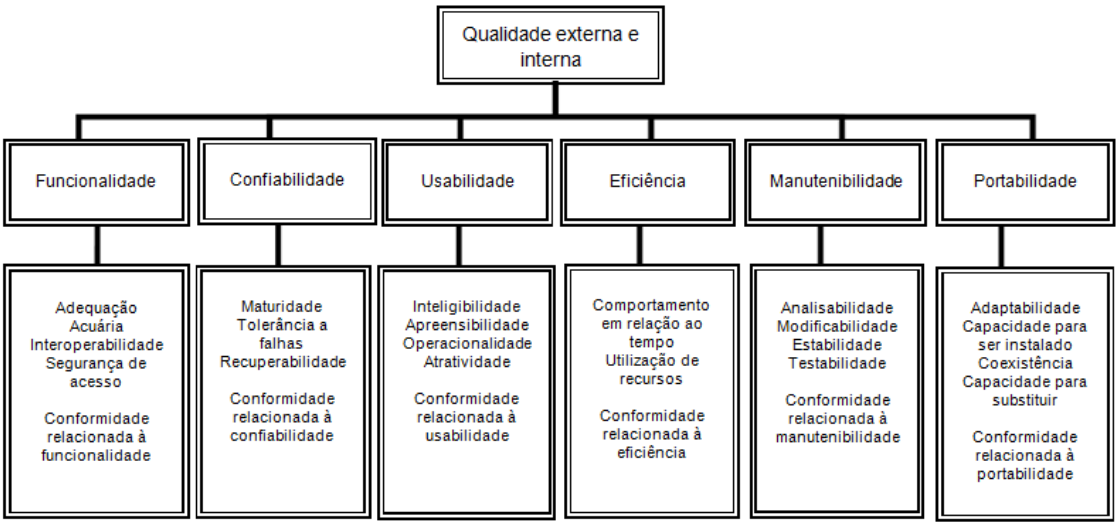


Figura 10 - Modelo de qualidade (*Modelo de Qualidade Externa e Interna ISO/IEC 9126. | Download Scientific Diagram*, 2019)

- **Funcionalidade** – capacidade de corresponder às funções e necessidades explícitas e implícitas do usuário quando o software é utlizado. Pode ser subdividido em:
    - Adequação: capacidade de fornecer um conjunto de funcionalidades adequadas aos objetivos do usuário.
    - Precisão: capacidade de fornecer funcionalidades com elevado nível de precisão.
    - Interoperabilidade: capacidade de interagir com outros sistemas.
    - Segurança: capacidade de proteger informação.

- **Confiabilidade**: capacidade de o software manter o nível de performance. Pode ser subdividido em:
    - Maturidade: capacidade de prevenir bugs no software.
    - Tolerância: capacidade de manter o sistema operacional apesar de possíveis falhas.

    - Recuperabilidade: capacidade de recuperar dados em caso de falha.

- **Usabilidade**: capacidade de o produto ser de fácil uso e compreensão, pode ser subdividido em:
    - Inteligibilidade: capacidade de o produto fazer o utilizador compreender se o produto é adequado.
    - Operacionalidade: capacidade do produto de ser de fácil compreensão e controlo.
    - Atratividade: capacidade de o produto ser atrativo para o utilizador.

- **Eficiência**: relacionamento entre o desempenho e os recursos utilizados, pode ser subdividido em:
    - Relação ao tempo: capacidade de fornecer tempos de resposta adequados:
    - Relação aos recursos: capacidade de utilizar quantidade de recursos adequados.

- **Manutenibilidade**: esforço necessário para efetuar alterações no software, pode ser subdividido em:
    - Analisabilidade: capacidade de diagnosticar problemas ou defeitos.
    - Modificabilidade: capacidade de efetuar facilmente alterações no produto.
    - Estabilidade: capacidade de evitar situações inesperadas causadas por modificações.
    - Testabilidade: capacidade de validar modificações.

- **Portabilidade**: capacidade que um produto tem em ser transferido entre ambientes, pode ser subdividido em:
    - Adaptabilidade: capacidade de ser adaptado entre diferentes ambientes sem necessitar de alterações.
    - Capacidade de instalação: capacidade de ser instalado em um específico ambiente.
    - Coexistência: capacidade que o produto tem de coexistir com outro produto, partilhando os mesmos recursos.
    - Capacidade de substituição: capacidade de um produto poder ser utilizado no lugar de outro produto, com o mesmo propósito, no mesmo ambiente.

Todas estas características podem ainda ser classificadas de acordo com a sua conformidade, isto é, a capacidade que o desenvolvedor tem de aderir a padrões e convenções de software relativas a cada um dos tópicos em questão.

2.7.4 *Padrões de desenvolvimento*

Uma das considerações mais cruciais no desenvolvimento de *software* é que este deve estar em constante mudança. Será continuamente atualizado, alterado e dotado de novas funcionalidades, aperfeiçoamentos e correções de bugs.

Por conseguinte, é fundamental criar um *software* passível de manutenção, ou seja, é fundamental construir um software que possa ser mantido por futuros desenvolvedores.

No entanto, é necessário compreender a definição de um "*software* que possa ser mantido por futuros desenvolvedores". Na prática, isto significa que qualquer desenvolvedor deve ser capaz de fazer alterações e modificações ao produto.

De forma a garantir o software possa ser mantido por futuros desenvolvedores existe inúmeros padrões de desenvolvimento, entre os quais os citados nos seguintes tópicos.

2.7.4.1 Onion Architecture

O padrão arquitetural "*Onion Architecture*" sugere que o software seja criado em camadas, cada uma com o seu próprio objetivo e respetivas responsabilidades, como é possível visualizar na seguinte figura:

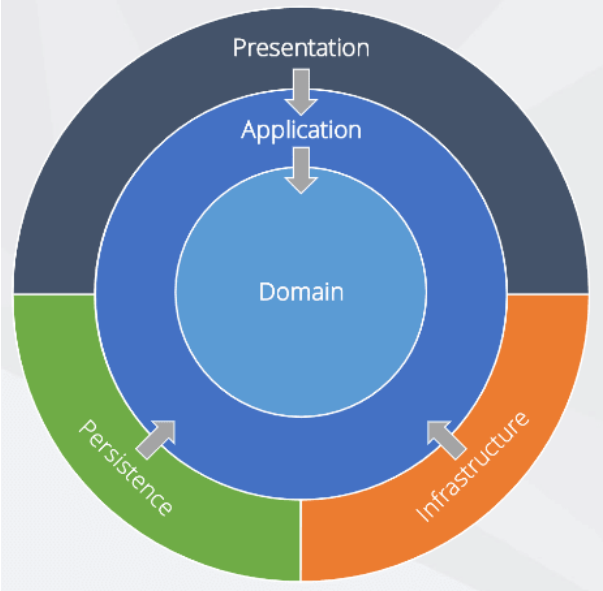


Figura 11 - Onion Architecture (Schaefer, 2020)

De forma a aplicar este padrão é necessário garantir que nenhuma camada interna conheça a lógica de uma camada mais externa. Isto aplica-se a todas as entidades do sistema, tais como funções, classes, variáveis etc. (Schaefer, 2020).

Este padrão é baseado em DDD (*Domain-Driven Design*) e pressupõe que cada serviço tenha 5 camadas distintas (se aplicáveis):

- ***Domain Layer*:** É a camada mais interna desta arquitetura. É nesta camada que são definidas regras de domínio e negócio, sendo uma camada completamente lógica que não deve incluir qualquer operação. Sendo a camada mais interna, esta não deve ter acesso a nada fora do seu âmbito.
- ***Application Layer*:** A camada de aplicação é a segunda camada mais interna desta arquitetura. Esta camada é aquela que utiliza a camada de domínio de forma a aplicar as regras de negócio por ela imposta. É nesta camada que se encontra a maioria dos serviços e interfaces.
- ***Repository Layer*:** Esta camada é uma das camadas mais externas desta arquitetura. É nesta camada que é tratada a lógica de persistência das entidades definidas da camada de domínio. Normalmente, apenas a lógica referente à base de dados deve estar presente nesta camada.
- ***Infrastucture Layer*:** Esta camada encontra-se ao mesmo nível da camada de repositório, ou seja, encontra-se na camada mais externa. Sendo assim, pode importar entidades das camadas Aplicação e Domínio e a estar ciente de tudo o que está incluído nas camadas internas. De uma forma geral, esta camada deve estar responsável por lógica de comunicação com APIs externas*, event listeners* etc.
- ***Presentation Layer*:** Esta camada encontra-se ao mesmo nível da camada de repositório e infraestrutura. É a camada responsável por expor os *endpoints* e pelas configurações da aplicação. Nesta camada não deve ser desenvolvida lógica de negócio, sendo que devem ser utilizadas as camadas de aplicação e domínio de forma a realizar as ações pretendidas.

2.7.4.2 Test-driven Development (TDD)

A prática de escrever testes como guia de desenvolvimento de software é conhecida como desenvolvimento orientado para testes (TDD) (Shemi, 2020).

Foi desenvolvido por Kent Beck no final dos anos 90 e baseia-se nos três seguintes passos:

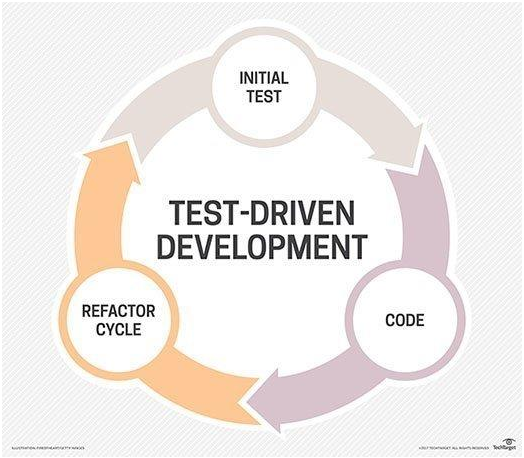


Figura 12 - Test-Driven Development (Shemi, 2020)

- Escrever um teste baseado na implementação que queremos fazer.
- Implementar a funcionalidade até que o teste passe.
- Melhorar o código do teste e da implementação de forma a torná-los bem estruturados.

Escrever primeiro o teste (*Test-First Programming*) têm principalmente 2 benefícios:

- É uma forma de implementar *self testing code*, uma vez que só se pode escrever algum código funcional após a implementação de um teste.
- Obriga o desenvolvedor a pensar primeiro no teste, forçando-o a pensar na funcionalidade de uma forma abstrata.

Apesar de ser um conceito simples de compreender, é comum cometerem-se erros constantes, entre os quais:

- Escrever demasiados testes de cada vez.
- Testes demasiado grandes e amplos.
- Apenas alguns elementos da equipa utilizarem TDD.
- Pouca manutenção dos testes.

3 Análise de valor

O principal objetivo da análise de valor é conseguir compreender como aumentar o valor de um artigo ou serviço tendo o menor custo possível e sem colocar em causa a sua qualidade (Nicola, n.d.-a).

Como referem David Hughes e Don Chafin: "Num ambiente tão acelerado, o desenvolvimento de produtos deve ser transformado num processo contínuo, iterativo e de aprendizagem concentrado no valor do cliente".

A Análise de Valor pode ser vista como um processo de análise e avaliação efetuado de forma formal e organizada. É também uma atividade de gestão que requer planeamento, controlo e coordenação.

Esta diz respeito à forma de como um produto para satisfazer as exigências de um cliente e o seu processo de revisão deve incluir uma compreensão da finalidade para a qual o produto é utilizado, tornando assim possível a compreensão do nível de encaixe que um produto pode possuir para determinado cliente ou consumidor.

A compreensão do objetivo do produto estudado é uma parte essencial deste procedimento (Rich and Holweg, 2000). Qualquer esforço para elevar o valor de um produto deve ter em conta dois fatores: o primeiro está relacionado com a utilização do produto, vulgarmente conhecido como *Use Value* (Valor de uso), que é responsável pela funcionalidade do mesmo.

O segundo fator está relacionado com a utilização do produto dada pelo cliente e é descrito por *Esteem value*, que especifica o valor que é atribuído pelo consumidor. Está normalmente relacionado com o valor estético e subjetivo.

3.1.1 *Processo de Inovação*

O processo de inovação corresponde à atividade de desenvolver ou alterar um produto de forma a adicionar valor ao mesmo (Koen, 2001).

Pode ser dividido em três áreas:

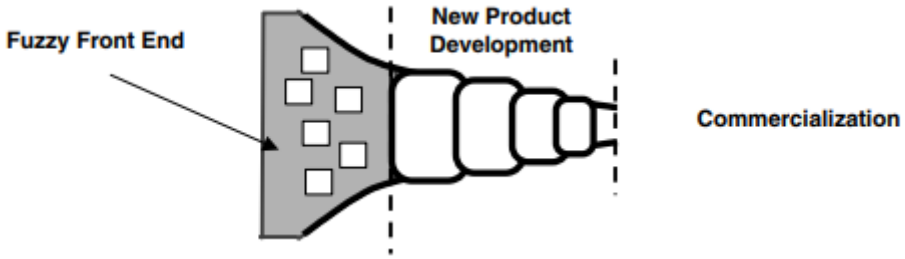


Figura 13 - Processo de inovação

Fuzzy Front End (FFE) corresponde à criação e um novo conceito/produto e a validação de um possível investimento nesse mesmo conceito/produto. O *New Product Development* (NPD) corresponde ao processo de desenvolvimento do produto idealizado e, por fim, o processo de comercialização.

Apesar da utilização corrente do FFE, não existe uma forma *standard* de o implementar, o que gera algumas dificuldades naquilo que é a utilização do mesmo. Deste pressuposto nasceu o *New Concept Development* (NCD), cujo objetivo é definir os elementos essenciais ligados ao *Fuzzy Front End* numa linguagem consistente (Koen, 2001).

O NCD consiste em 3 partes distintas:

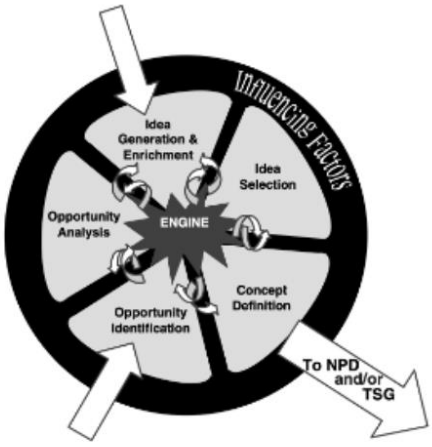


Figura 14 - New Concept Development

- O motor ou “bullseye” representa a liderança, a cultura e a estratégica de negócio da organização, fatores estes que influenciam os cinco elementos-chave que são controláveis pela organização.

- A área do raio interno define os cinco elementos-chave controláveis: identificação de oportunidades, análise de oportunidades, geração e enriquecimento de ideias, seleção de ideias e definição do conceito.

- Os fatores de influência consistem em fatores externos como canais de distribuição, política governamental, clientes, concorrentes, clima político-económico e a evolução das ciências. Estes fatores podem afetar todo o processo de inovação até à sua comercialização e são incontroláveis para a organização.

3.1.2 *Identificação de oportunidades*

A identificação de oportunidades é o elemento que define as oportunidades que uma determinada organização pode ou não aproveitar.

Pode ser definida como uma direção totalmente nova para o negócio, um novo produto ou serviço ou até mesmo uma melhoria em um produto ou serviço existente (Koen, 2001).

A identificação de oportunidades pode surgir de uma única pessoa que reconheça uma necessidade não satisfeita do cliente ou um problema anteriormente não detetado. Pode também permitir ganhar novas noções de necessidades do mercado que não eram previamente conhecidas.

Com a compra efetuada pela Maersk e consequente mudança de contexto profissional, surgiu a oportunidade de, para além de migrara plataforma *de e-fulfillment*, atualizá-la e restruturá-la. Esta migração, para uma arquitetura orientada a microsserviços é uma transformação esperada, no sentido em que é uma tendência no universo da tecnologia. É uma arquitetura vista como superior em comparação à monolítica apesar de, muitas vezes, envolver uma migração com elevada complexidade.

Desta forma, é identificada como uma oportunidade, que apesar de adicionar uma complexidade considerável, pode trazer inúmeros benefícios, como por exemplo, uma maior escalabilidade, flexibilidade e independência entre equipas.

3.1.3 *Análise de oportunidades*

Após a identificação de oportunidades, analisam-se as mesmas de forma a determinar e compreender se esta pode ou não ter valor para o contexto atual.

Isto implica fazer avaliações precoces e muitas vezes incertas da tecnologia e do mercado e envolver um esforço extensivo para atingir grupos-alvo, estudos de mercado e/ou experiências científicas (Koen, 2001).

A capacidade empresarial e a competência são avaliadas nesta etapa, onde será determinado o caminho a seguir. A efetividade das equipas responsáveis pela análise de oportunidades depende do tamanho, âmbito e cultura da organização.

Posto isto, é necessário analisar a oportunidade identificada anteriormente: atualização e restruturação de uma plataforma através de migração para uma arquitetura orientada a microsserviços.

Em uma análise realizada pela O'Reilly em 2020 (*Microservices Adoption in 2020 – O'Reilly*, 2020), é possível compreender que apenas aproximadamente 23% das empresas alvo do estudo utilizam uma arquitetura monolítica, enquanto aproximadamente 77% das empresas alvo utilizam uma arquitetura baseada em microsserviços, como é possível visualizar no seguinte gráfico:

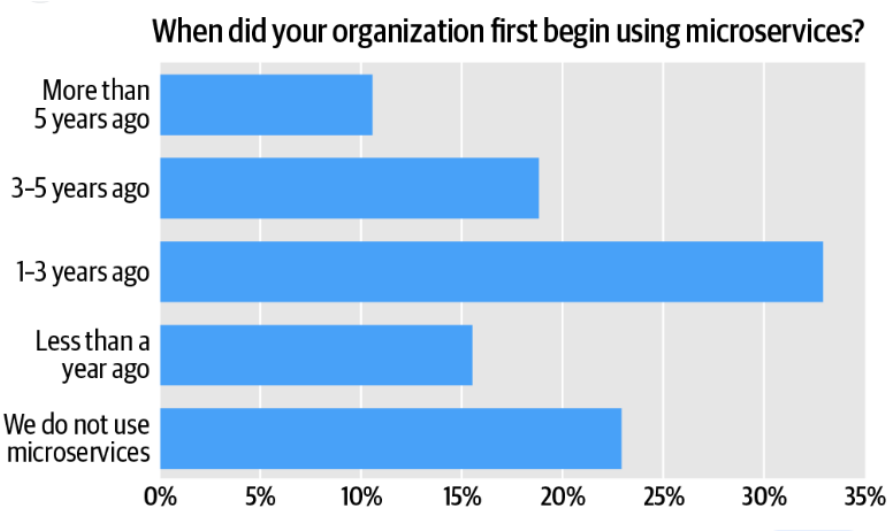


Figura 15 - Gráfico de análise à data de adoção de microsserviços

Para além disso, é possível verificar que aproximadamente 49% das empresas alvo começaram a utilizar uma arquitetura orientada a microsserviços entre 2017-2020, facto este que revela um crescimento contínuo na utilização desta mesma tecnologia.

Ainda no mesmo estudo foram listadas as principais vantagens da utilização desta arquitetura descritas por essas mesmas empresas:

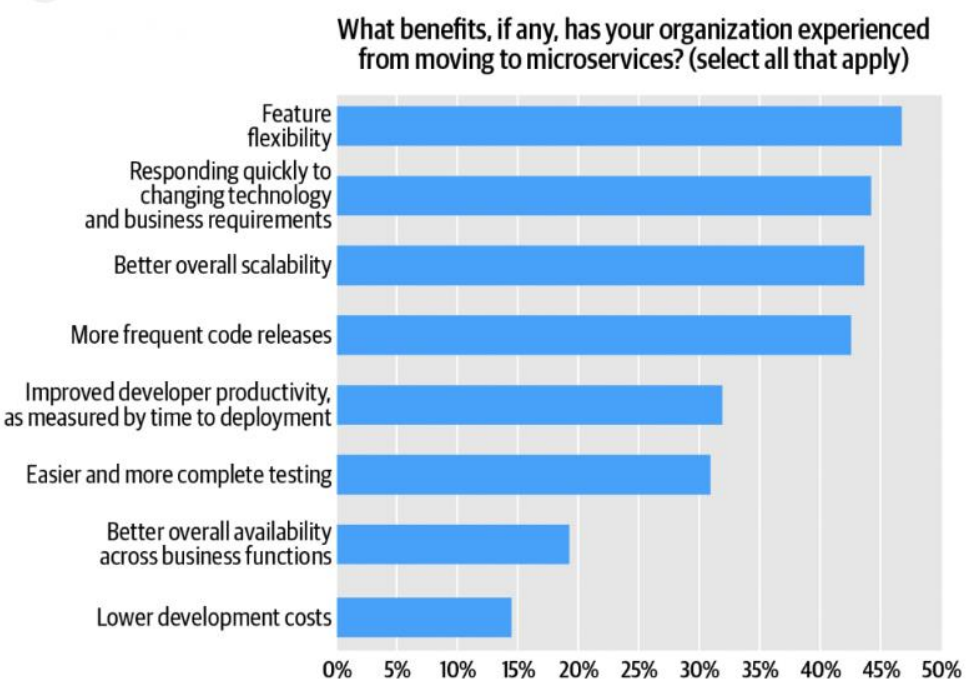


Figura 16 - Gráfico de análise às vantagens proporcionadas pelos microsserviços

Entre as vantagens destacadas, são de realçar a flexibilidade, manutenibilidade e escalabilidade, referidas quer no desenvolvimento, quer nas mudanças tecnológicas ou mesmo a nível de requerimentos. É de realçar também o facto de que apenas aproximadamente 14% das empresas referiram um custo menor no desenvolvimento, provavelmente referente ao custo da infraestrutura, visto que é conhecido que uma arquitetura orientada a microsserviços tem mais custos associados a esse requisito.

Para finalizar e, ainda referente ao mesmo estudo, é possível constatar o seguinte:

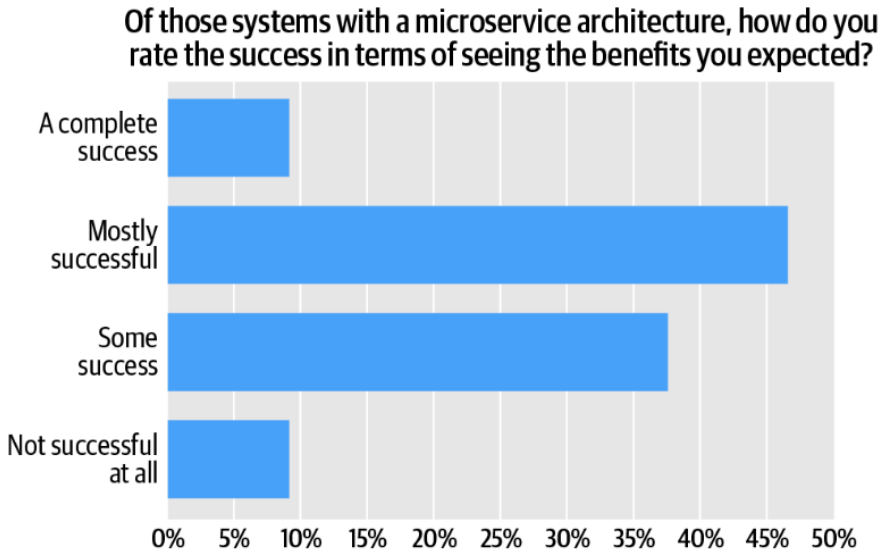


Figura 17 - Gráfico de análise ao sucesso da adoção de microsserviços

Pode-se verificar que apenas aproximadamente 8% das empresas classificaram a sua adoção de uma arquitetura baseada em microsserviços como um fracasso. Esta estatística é um ponto positivo para a análise referente à oportunidade identificada.

Com base nesta análise mercado pode-se concluir que a oportunidade identificada é valida para o contexto atual e pode representar uma mais-valia para o projeto em questão.

3.1.4 *Geração e enriquecimento de ideias*

A geração e enriquecimento de ideias pode ser descrito como um processo formal, incluindo sessões de brainstorming de modo a promover a geração de ideias novas ou renovar ideias antigas para a oportunidade identificada (Koen, 2001). Para além de processos formais de geração e enriquecimentos de ideias, novas ideias podem também surgir em processos não formais presentes no quotidiano dos envolvidos.

As ideias de seguida apresentadas têm origem em inúmeras reuniões formais e conversas informais:

- **Estudo de diferentes abordagens**: Com base nesta ideia, devem ser identificados e estudados diferentes padrões de migração e diferentes arquiteturas orientadas a microsserviços.

- **Resolução da dívida técnica**: Com base nesta ideia, deve ser resolvida a dívida técnica associada ao sistema atual. A resolução da dívida técnica pode ser atingida através da adoção de variados padrões de qualidade.

- **Uniformização de serviços**: Com base nesta ideia, deve ser elaborado um *template* nos quais os desenvolvedores se devem basear para a construção de novos serviços. Desta forma, assegura-se que as estruturas, boas práticas, convenções e padrões a utilizar são uniformes em diferentes serviços.
- **Melhorias no processo de desenvolvimento:** Com base nesta ideia, pretende-se que através da aplicação das métricas de qualidade citadas no capítulo de estado da arte, ocorra um crescimento no número de *story points* entregues por sprint.

3.1.5 *Seleção de ideias*

A seleção pode ser tão simples como a escolha individual entre muitas opções geradas independentemente da complexidade associada a qualquer uma das ideias (Koen, 2001).

A maioria da seleção de ideias envolve uma série iterativa de fases que provavelmente incluem múltiplas passagens através da identificação de oportunidades, análise de oportunidades, e geração e enriquecimento de ideias.

Na maioria dos casos, a dificuldade passa por decidir entre múltiplas ideias e não propriamente gerar as ideias. A seleção de uma ideia é crítica para o futuro e para o alcance objetivo proposto.

A seleção de ideias começa frequentemente com uma análise individual inicial, com pouco mais do que a própria ideia a considerar. Sem um processo de decisão formal, a maioria das novas ideias desaparece numa espécie de "buraco negro".

No contexto deste projeto, tanto a geração e enriquecimento de ideias como a seleção das mesmas são da competência de todos os elementos envolvidos no mesmo, como já referido na secção anterior.

Desta forma, todas as ideias mencionadas na respetiva secção serão implementadas.

3.1.6 *Definição de conceito*

A definição do conceito é o elemento final do *New Concept Development*. É na definição de conceito que se deve fazer um "caso convincente" para o conceito de negócio ou proposta tecnológica em questão.

A oportunidade identificada consiste em atualizar, restruturar e migrar uma plataforma de *e-fulfillment* através do desenvolvimento de uma estratégia de migração e do desenvolvimento de uma arquitetura orientada a microsserviços. Tendo em conta que, com base nos dados citados na análise de oportunidades, a oportunidade identificada é valida e pode, de facto, constituir valor para o decorrer do projeto e que, o desenvolvimento da nova plataforma global de *e-fulfillment* é um projeto na qual a organização acredita, podemos afirmar que o desenvolvimento deste projeto pode ser o primeiro passo para o grande objetivo da Maersk: "tornar-se, não um, mas o integrador global de cadeias de suprimentos, oferecendo soluções de logísticas que se conectam e simplificam todo o processo dos seus clientes".

3.2 Valor

Valor refere-se ao benefício ou vantagem que um produto ou serviço proporciona ao cliente, visto do ponto de vista do mesmo. Normalmente está associado a benefícios funcionais, pois referem-se a problemas específicos que necessitam de ser solucionados.

Perceived value pode ser definido como a relação entre aquilo que o cliente está disposto a dar e aquilo que o cliente espera receber. Sendo assim, o *perceived value* é uma definição intimamente ligada ao ponto de vista do cliente e pode variar de cliente para cliente. Se aquilo que o cliente espera receber têm mais valor do que aquilo que ele está disposto a dar, pode-se afirmar que a solução tem, de facto, valor para o cliente. Caso as condições estejam invertidas, a solução não tem valor.

De forma a avaliar o *value for the customer* foi desenvolvida a seguinte tabela que relaciona os *benefits* (aquilo que o cliente espera receber) e os *sacrifices* (aquilo que dá em troca):

Tabela 1 - Benefícios e Sacrifícios

Benefits	**Sacrifices**
Simplificação de processos Total visibilidade sobre processos Análise de dados Baixo custo	Custo associado à utilização da plataforma Perda parcial do controlo sobre armazenamento e distribuição Adaptação ao novo modo de trabalho

3.3 Value proposition

A *value proposition* ou proposta de valor tem como principal objetivo demonstrar o valor que um produto ou serviço fornecerá aos seus clientes através da descrição dos benefícios dados pelo produto ou serviço e como este aliviará as suas necessidades específicas ou problemas identificados.

A proposta de valor deve ser clara, concisa e facilmente compreendida pelos clientes-alvo. A seguinte imagem pretende demonstrar a proposta de valor do projeto abordado neste documento com base no modelo de Osterwalder:

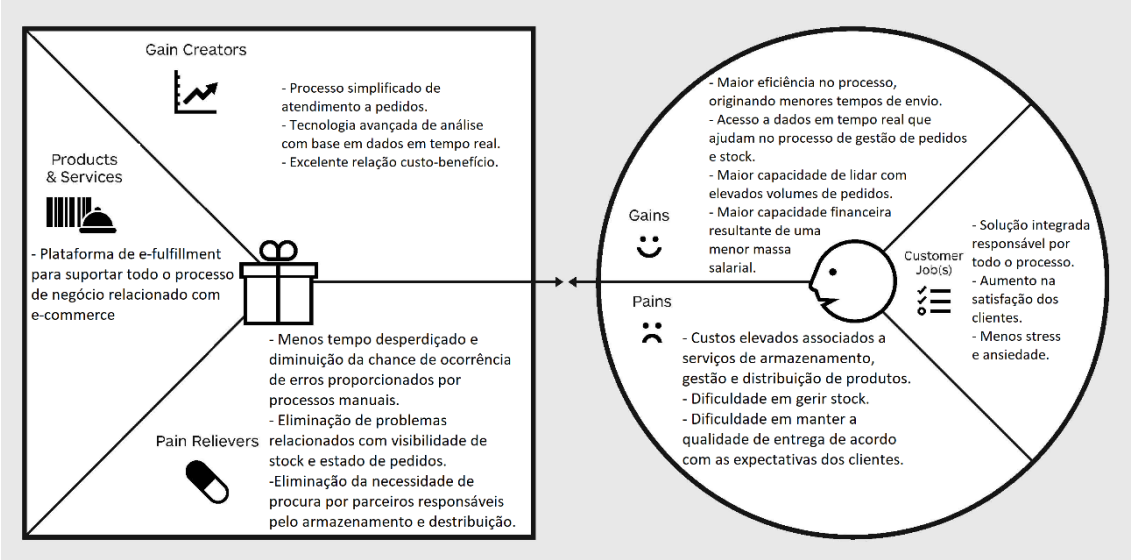


Figura 18 - Value proposition baseada no modelo de Osterwalder

4 Análise e Conceção

Nesta secção é analisado, ilustrado e descrito a atualidade do sistema. As ilustrações são baseadas em diagramas UML e estarão também anexadas no final do documento de forma a proporcionar uma melhor visualização das mesmas.

Após uma apresentação do sistema atual, são efetuadas análises a duas diferentes arquiteturas resultantes da segmentação do mesmo. A escolha final em relação à arquitetura a utilizar será efetuada com base no método multicritério AHP.

Finalmente, o enunciar de não só os requisitos funcionais identificados de forma a construir um serviço capaz de suportar os processos atuais, mas também dos requisitos não funcionais.

4.1 Visão geral da atualidade do sistema

Nesta secção são apresentados individualmente os diferentes módulos que compõe a plataforma de *e-fulfillment* atual. Também é especificada a forma de como interagem entre si e a sua relação com componentes externos.

Apesar do foco do projeto estar sobre o módulo de OMS, a análise do sistema como um todo é importante não só para contextualização, mas também para a compreensão de conceitos e processos associados à área de negócio em questão. Em adição, é importante referir que, à data de início deste projeto, o autor deste documento não possuía qualquer tipo de conhecimento relativo ao sistema.

A granularidade das vistas apresentadas vai aumentando de forma a demonstrar da melhor forma possível a atualidade do sistema.

4.1.1 *E-Fulfillment*

A seguinte figura pretende demonstrar a plataforma atual, assim como alguns dos módulos que auxiliam a comunicação com serviços externos:

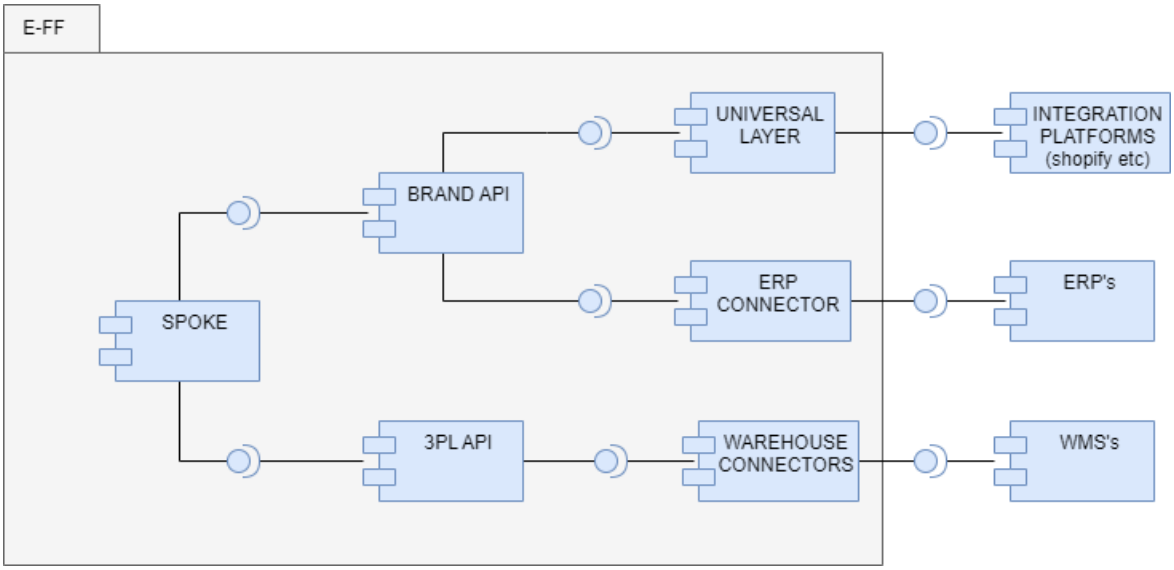


Figura 19 - E-fulfillment diagrama de componentes

O Spoke é o componente que agrega toda a lógica de negócio associada ao *e-fullfilment* e é sobre este que incide grande parte do foco deste documento.

Como é possível visualizar, o Spoke apenas comunica diretamente com dois módulos (BRAND API e 3PL API), módulos estes que fazem parte de uma camada pública que têm como principal objetivo a interação com serviços externos. A comunicação com estes serviços externos é realizada via HTTP.

No caso das plataformas de vendas online (shopify, woocomerce etc.), esta comunicação é feita via um módulo denominado "*Universal Layer*". Este serviço permite às marcas que possuam os seus produtos disponíveis em uma plataforma online integrar-se com o sistema de *e-fulfillment* de forma automática.

Os sistemas integrados de gestão empresarial (ERPs) também são integráveis com o sistema através do módulo "ERP connector". Este serviço é responsável por comunicar com qualquer ERP configurado pela marca.

Por último, é necessário também comunicar com os sistemas de gestão de armazéns. O serviço de 3PL, abreviatura para *third-party logistics,* é responsável por agregar toda a lógica de negócio comum a todos os armazéns e pela comunicação aos conetores. Cada armazém terá o seu *conector* específico de forma a satisfazer as suas necessidades.

4.1.2 *Spoke*

A seguinte figura pretende representar a constituição atual do Spoke, que consiste no *core* da plataforma atual. Os módulos representados na seguinte figura são também analisados e detalhados nas secções seguintes.

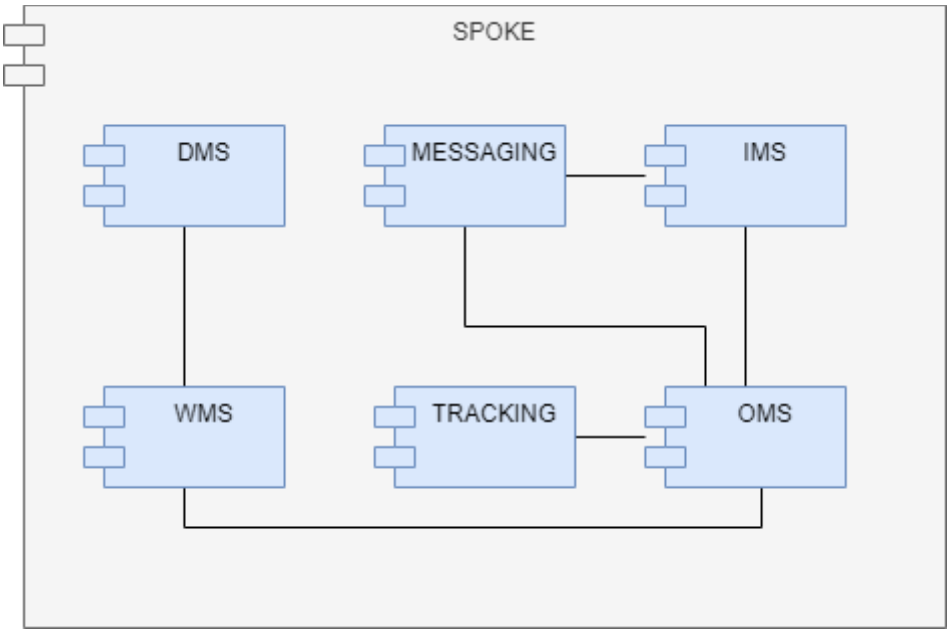


Figura 20 - Spoke representação modular

Este sistema foi desenhado e pensado como um *modular monolith*, isto é, constituído por diferentes submódulos. Um *modular monolith* apresenta algumas das vantagens também presentes em uma arquitetura orientada a microsserviços, visto que estes são independentes e, como tal, permitem uma maior capacidade de manutenção. No entanto e, apesar da divisão modular, o sistema é *deployed* em um só artefato.

Apesar de ter sido desenhado como tal, a verdade é que, com o acumular de diferentes desenvolvimentos e requisitos, o nível de acoplamento entre diferentes módulos foi aumentando e foi-se perdendo aquilo que seria a ideia inicial, isto é, módulos independentes. O acoplamento identificado não é alarmante visto que, na sua maioria, é referente a validações entre diferentes entidades, algo comum mesmo em arquiteturas baseadas em microsserviços. Mesmo assim, foram também identificadas execuções de serviços presentes noutros módulos, resultando em uma dívida técnica a ser mitigada.

As próximas secções demonstram e analisam cada um dos módulos acima representados de forma a obter-se uma melhor percepção de como o sistema atual é composto e as diferentes relações entre diferentes módulos.

4.1.3 *Delivery Management System (DMS)*

A seguinte figura representa o modelo de domínio e as relações entre as diferentes entidades deste módulo.

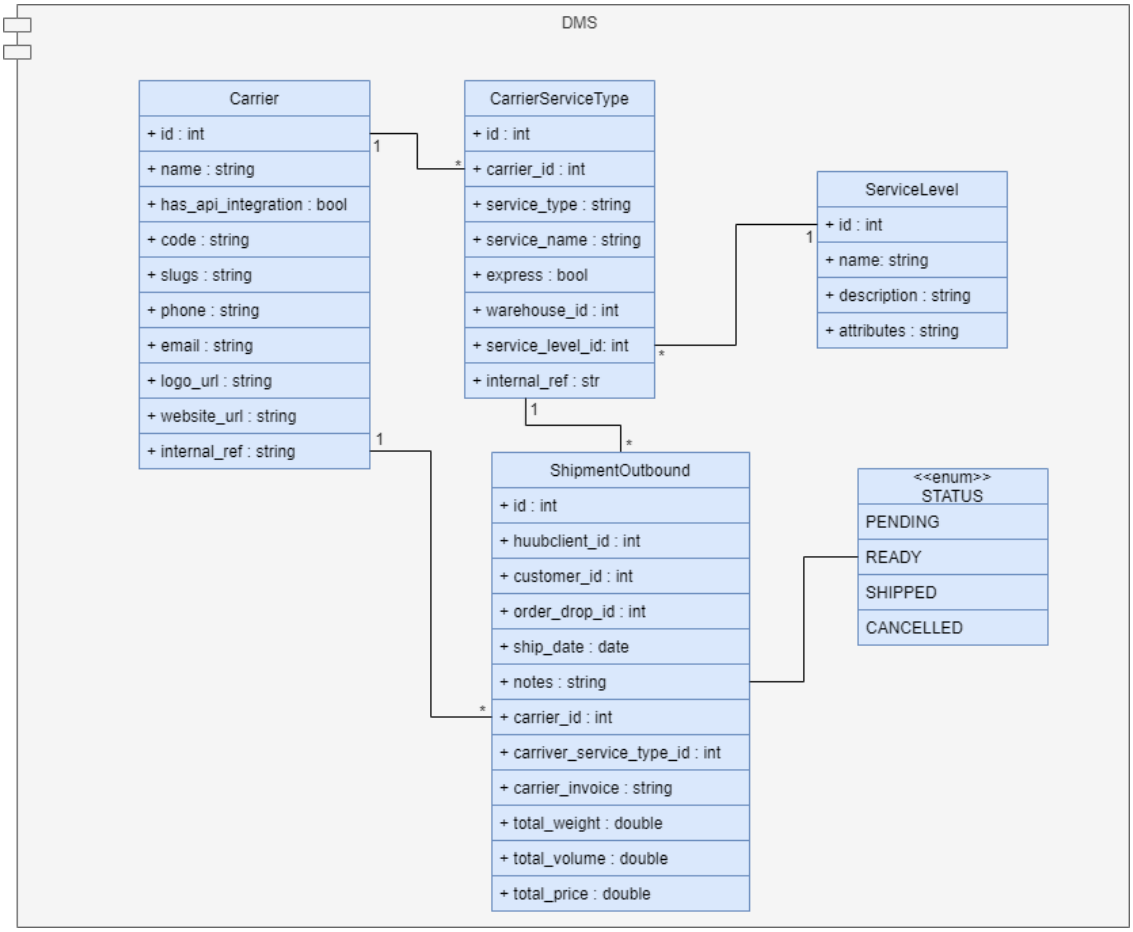


Figura 21 - DMS Modelo de domínio

DMS é o modulo responsável pelas entidades relativas à comunicação com os *Carriers*, ou seja, responsável pela comunicação com entidades responsáveis pela distribuição de produtos.

A entidade "Carrier" é responsável por definir os diferentes distribuidores (e.g CTT, DHL etc.). É importante referir que este módulo está diretamente anexado à antiga realidade da HUUB, que não possuía distribuidores próprios. Com a mudança de contexto e, tendo em conta que a Maersk possui formas de distribuição independente, será um módulo que sofrerá inevitáveis alterações.

Os distribuidores possuem também um tipo de serviço, *CarrierServiceType*, e *ShipmentOutbounds*, que correspondem ao produto, ou conjunto deles, que o distribuidor está responsável por distribuir.

4.1.4 *Inventory Management System (IMS)*

A seguinte figura representa o modelo de domínio e as relações entre as diferentes entidades deste módulo.

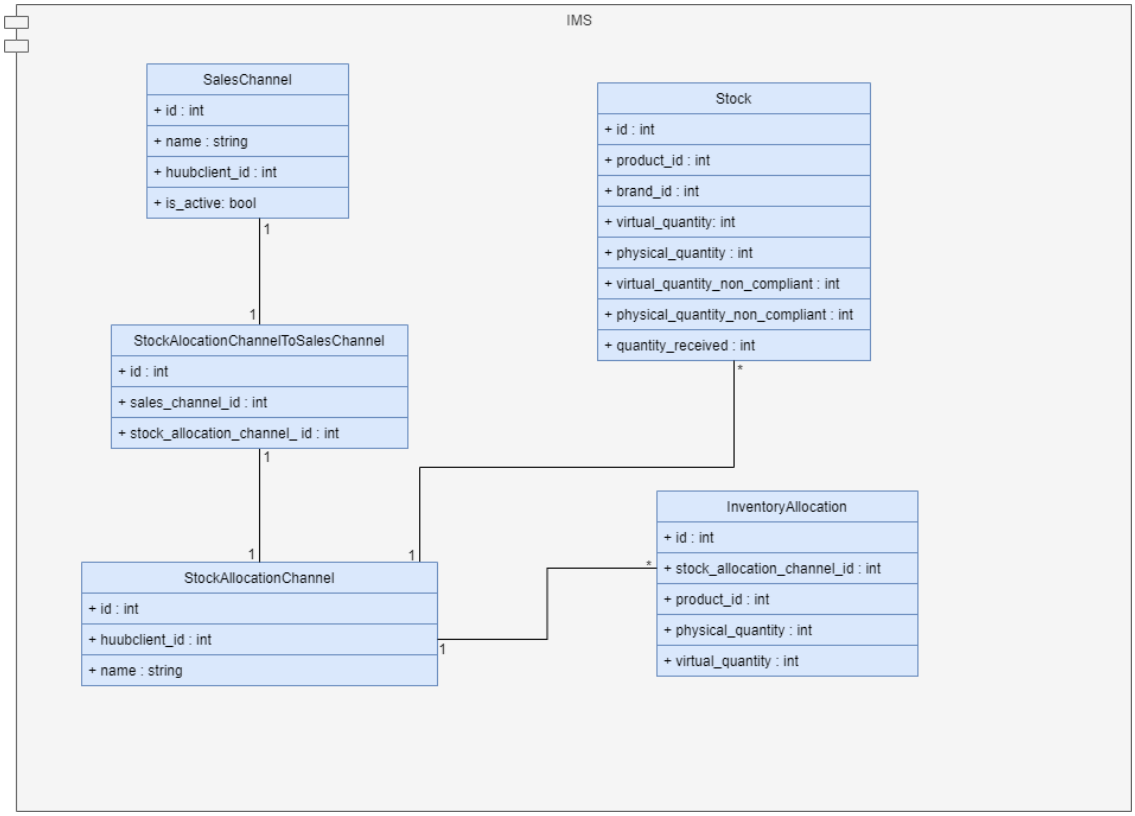


Figura 22 - IMS Modelo de domínio

IMS é o modulo responsável por gerir e controlar o stock das diferentes marcas existentes no sistema.

Atualmente a gestão é feita por canais de venda (*SalesChannel*). Esta entidade é responsável pela divisão de stock por diferentes canais, visto que o sistema atual não suporta apenas e-commerce, mas também venda física.

Estes canais são, por defeito, três por marca: online, físico e partilhado. Desta forma as marcas conseguem, por exemplo, ter visibilidade de quando um determinado produto está em falta, conseguindo identificar automaticamente qual o canal de venda em questão e, rapidamente, solucionar o problema.

É um módulo essencial pois é ele que permite toda a gestão de quantidades de produtos.

4.1.5 *Order Management System (OMS)*

A seguinte figura representa o modelo de domínio e as relações entre as diferentes entidades deste módulo.

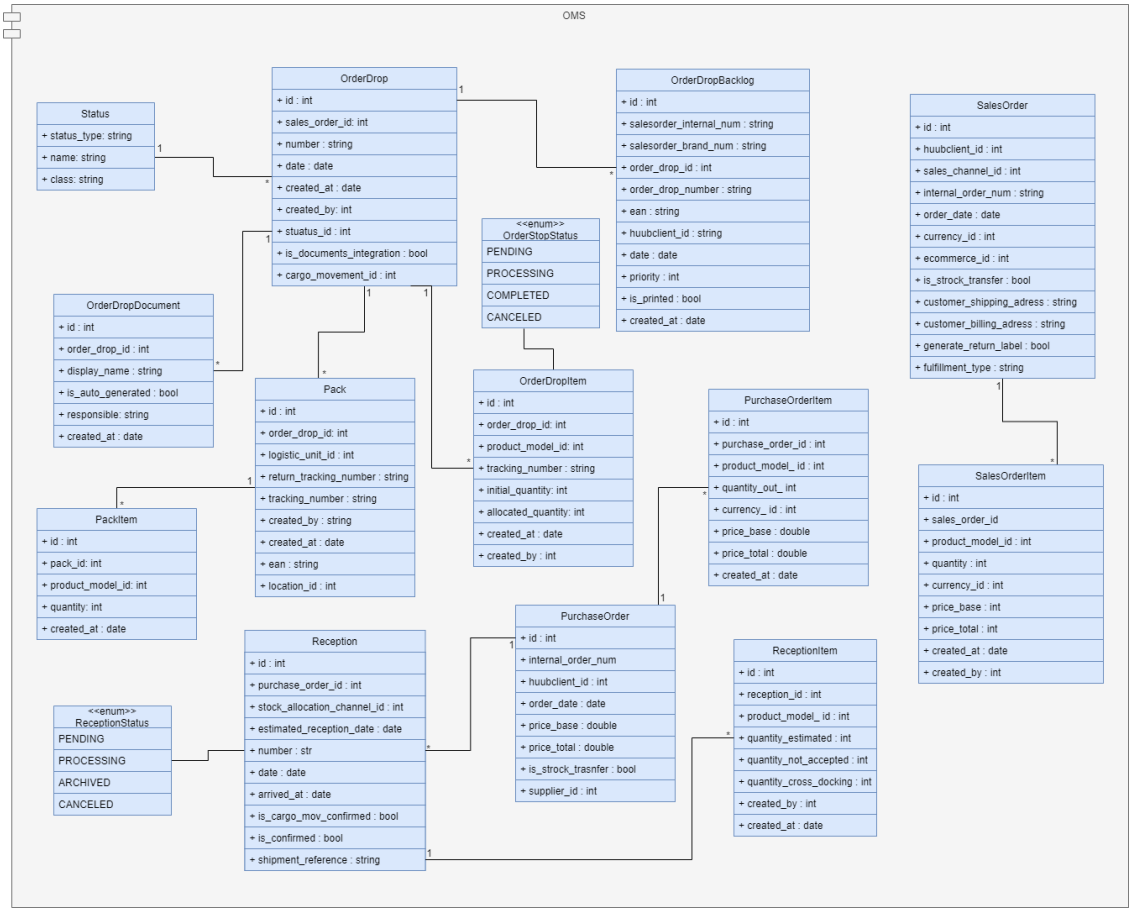


Figura 23 - OMS Modelo de domínio

OMS é o maior e mais complexo modulo do Spoke, sendo também neste que incidirá o maior foco deste documento.

É responsável pela gestão da entidade *OrderDrop*, *PurchaseOrder* e *SalesOrder*.

A entidade *PurchaseOrder* é responsável por o acréscimo da quantidade de um ou mais produtos em um determinado armazém. É uma "ordem de compra" e é ação efetuada pelas marcas quando um mais produto está em falta em um armazém específico. Uma PurchaseOrder é representada por uma ou mais *Receptions*, ou seja, uma *PurchaseOrder* não tem necessariamente que ser completa em um único ato de entrega.

A entidade *SalesOrder* representa uma venda da marca. É uma "ordem de venda" e esta é, normalmente, comunicada via serviço externo (e.g shopify) e pretende dar visibilidade às vendas que estão a ser efetuadas, assim como ajustar o stock.

4.1.6 *Warehouse Management System (WMS)*

A seguinte figura representa o modelo de domínio e as relações entre as diferentes entidades deste módulo.

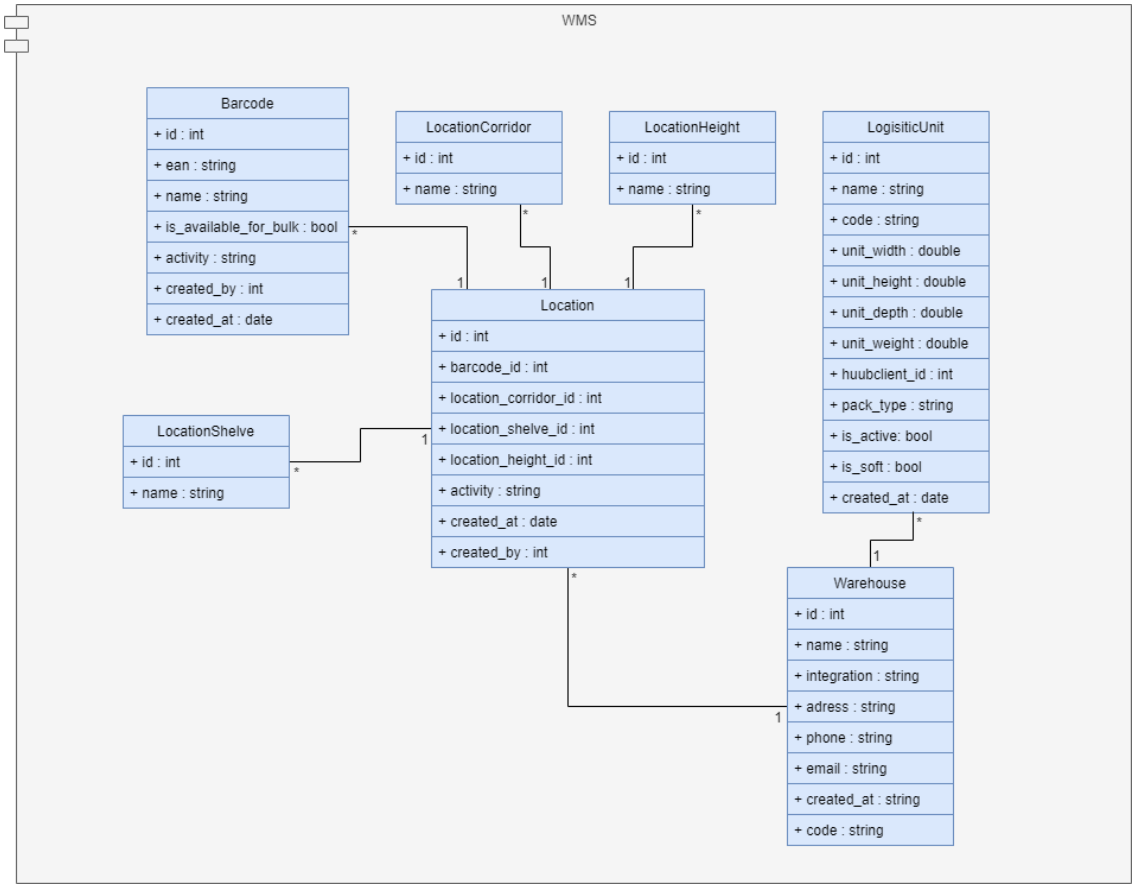


Figura 24 - WMS Modelo de domínio

WMS é o modulo responsável pelas configurações de armazém.

É o módulo menos utilizado visto que o seu objetivo passa por informar os responsáveis de armazém da localização de um terminado produto no interior do armazém. Esta funcionalidade pressupõe uma enorme colaboração e organização por parte das redes de armazéns, algo que nem sempre acontece.

Este módulo está também ligado à antiga forma de funcionamento da HUUB e é um sério candidato a ser completamente descartado no novo sistema.

4.1.7 *Tracking*

A seguinte figura representa o modelo de domínio e as relações entre as diferentes entidades deste módulo.

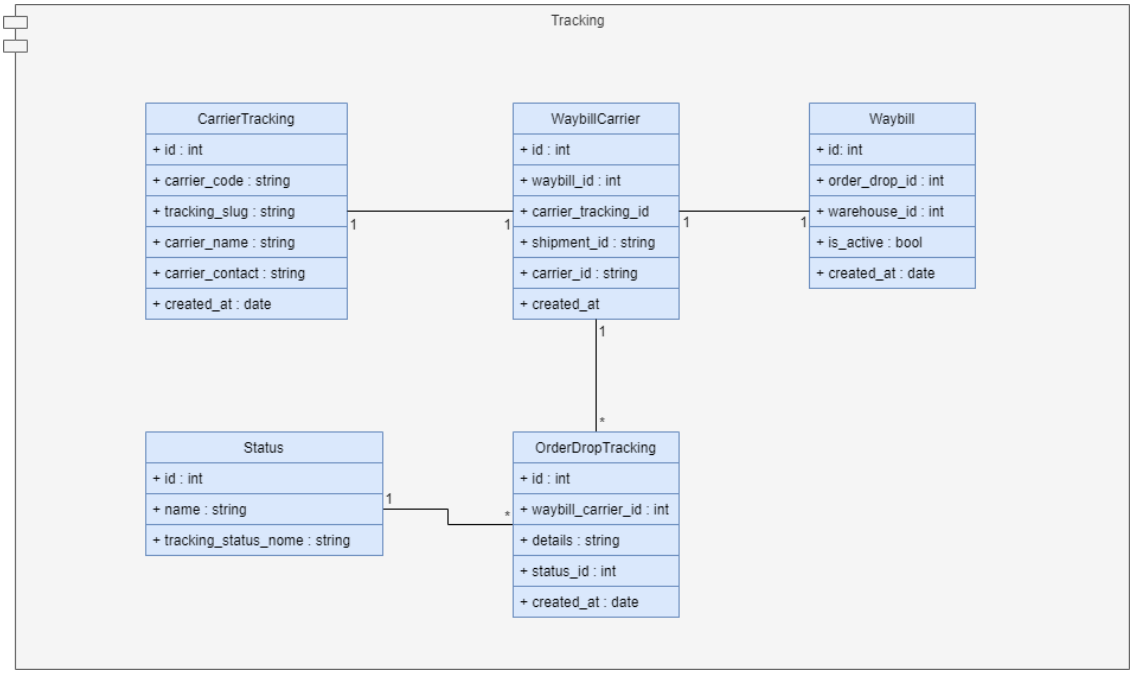


Figura 25 - Tracking Modelo de domínio

O módulo de *Tracking* pretende dar às marcas a possibilidade de, em real time, saber a localização de todos os produtos, desde os que estão em armazém, aos que estão em movimento, seja para um outro armazém, seja para o cliente final.

É, naturalmente, um módulo candidato a ser descartado no sistema atual visto que a Maersk já possui sistemas de *tracking.*

4.2 Abordagem de Segmentação

Nesta secção é abordada a forma de como se pretende segmentar o sistema atual, utilizando DDD de forma a compreender e desenhar os diferentes microsserviços.

O objetivo passa obter uma segmentação capaz de tornas os novos serviços o mais independentes possível, segmentando totalmente o monólito atual.

Desta forma, propõe-se as seguintes arquiteturas para a nova plataforma ECL:

4.2.1 *Segmentação por módulos*

A primeira abordagem identificada tem como objetivo utilizar os módulos já existentes e reaproveitar a segregação já identificada para criar diferentes microsserviços.

Este reaproveitamento envolve, naturalmente, uma análise de forma a mitigar uma das dívidas técnicas identificadas: a execução de serviços existentes em outros módulos. Este desacoplamento será atingido através da aplicação de um *message broker*, que foi pensado e concebido exatamente para resolver estes problemas de acoplamento.

De forma a atingir esta divisão são utilizados conceitos/técnicas/estratégias enunciadas no respetivo capítulo no Estado da Arte deste mesmo documento.

Consequentemente, obtêm-se a seguinte arquitetura:

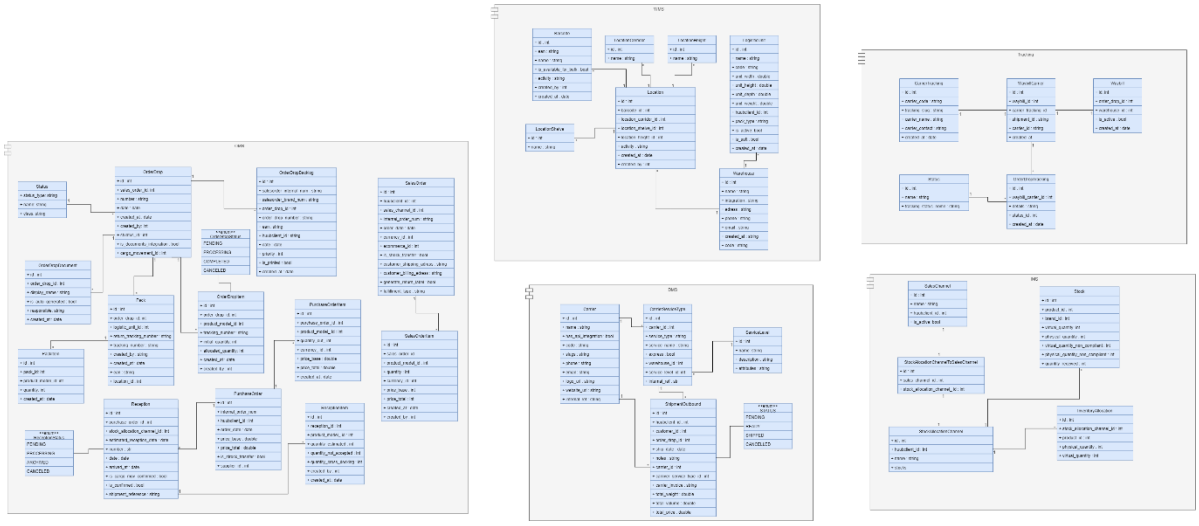


Figura 26 - Primeira proposta arquitetural

Esta solução teria um menor período de implementação, pois é aquela que menos alterações pressupõe. Desta forma seria possível, teoricamente, aplicar o *Strangler Pattern* como padrão de migração de forma direta e objetiva visto que um módulo dentro do “Spoke” teria uma relação de um para um com um novo microsserviço desenvolvido.

4.2.2 *Segmentação por bounded context*

Esta solução tem como objetivo dividir os módulos já existentes através da identificação de diferentes *bounded contexts*.

Um *bounded context* é um conceito essencial em DDD e refere-se a uma área distinta, normalmente delineada por relações entre entidades. Pode, em alguns casos, ser pensado como um subdomínio dentro de um domínio já identificado.

O primeiro passo é, através da análise, identificar quais são os *bounded contexts* existentes nos módulos que constituem o sistema atual. Através dessa mesma análise compreendeu-se que, à excepção do módulo OMS, todos os módulos apresentam um único *bounded context*, que inclui todas as entidades apresentadas nas figuras representativas de cada módulo.

A seguinte figura representa o módulo OMS, onde é possível também visualizar os diferentes *bounded contexts* identificados (delineados a diferentes cores):

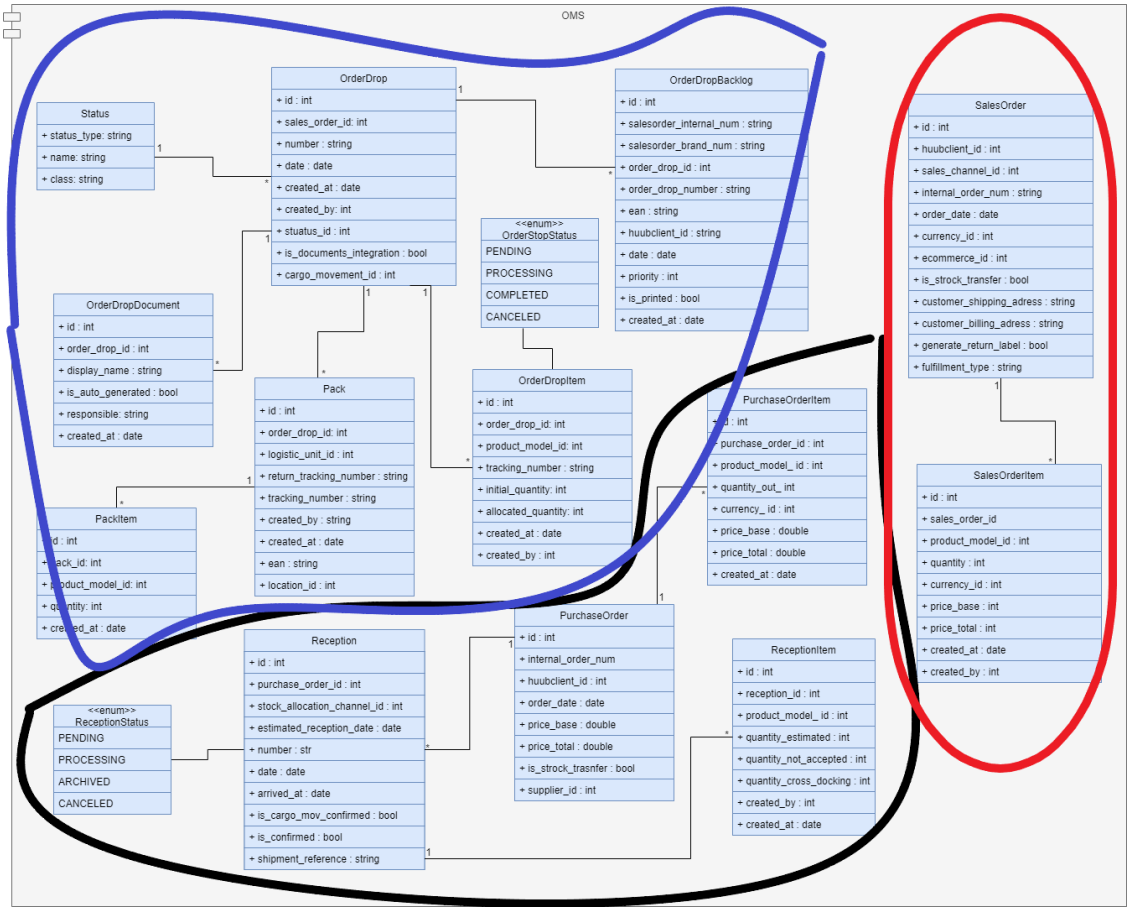


Figura 27 - OMS Bounded contexts

Com base nesta divisão, é possível compreender que o módulo OMS possui três *bounded contexts*, onde cada um possui um objeto central, sendo eles: "*OrderDrop*" (delineado a azul), "*SalesOrder*" (delineado a vermelho) e "*PurchaseOrder*" (delineado a preto).

Estes *bounded contexts* não apresentam qualquer tipo de relação entre si e correspondem a processos completamente diferentes, sendo que, encontram-se localizados no mesmo módulo pelo simples facto de, na sua nomenclatura, constar a palavra "*order*".

Com o resultado desta segmentação e, compreendendo que cada um destes *bounded context* daria origem a um microsserviço, foi tomada a decisão de iniciar o processo de migração deste módulo a partir do *bounded countext* cujo objecto central é a "*PurchaseOrder*". Esta decisão

está diretamente ligada ao plano traçado pela empresa, sendo desta forma completamente alheia ao autor do documento.

Consequentemente, propõe-se a seguinte arquitetura:

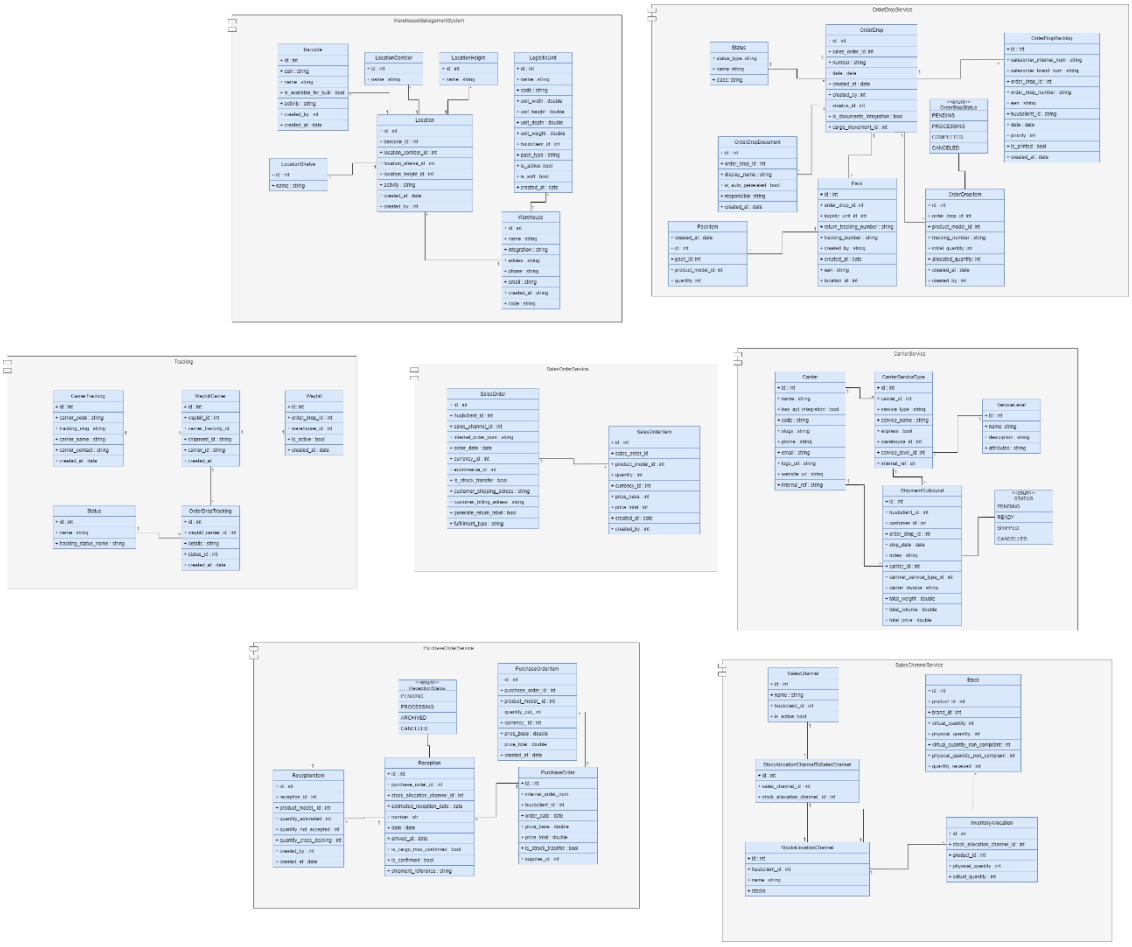


Figura 28 - Segunda proposta arquitetural

4.3 Métodos Multicritério

Métodos de decisão multicritério tem como objetivo auxiliar a decisão entre um conjunto de alternativas previamente definidas (Nicola, n.d.-b). No contexto deste projeto existem decisões que necessitam ser tomadas antes do começo da implementação, nomeadamente o estilo arquitetural a utilizar. No capítulo anterior foram sugeridas duas possíveis arquiteturas resultantes da segmentação do monolito descrito e, sendo assim, há uma necessidade de tomar a decisão sobre qual arquitetura adotar. Nesta secção será utilizado o modelo AHP como ferramenta de auxílio à tomada de decisão.

4.3.1 *Analytic Hierarchy Process (AHP)*

Analytic Hierarchy Process, ou Método de Análise Hierárquica, é um modelo criado pelo professor Thoma L. Saaty, em 1980.

É um método que utiliza critérios quantitativos e qualitativos e tem como principal objetivo a divisão do problema em diferentes níveis hierárquicos de forma a facilitar a tomada de decisão.

O primeiro passo na aplicação do método AHP clássico consiste na definição dos diferentes critérios pelos quais as diferentes alternativas são avaliadas de forma a poder construir uma árvore hierárquica de decisão (Nicola, n.d.-b).

Para o contexto do projeto atual serão utilizados os seguintes critérios definidos em colaboração com a equipa de desenvolvimento:

- **Escalabilidade**: Refere-se à facilidade de escalamento de forma a satisfazer as necessidades atuais do sistema.
- **Manutenibilidade**: Refere-se à facilidade com que a arquitetura pode ser atualizada e mantida a longo prazo.
- **Flexibilidade**: Refere-se á facilidade com que a arquitetura pode suportar, por exemplo, novos requisitos e regras de negócio.
- **Interoperabilidade**: Refere-se à facilidade com que a arquitetura interage com outros sistemas e serviços.

De forma a facilitar a compreensão da árvore hierárquica de decisão, são dados nomes representativos das diferentes arquiteturas sugeridas:

- **Arquitetura A:** Arquitetura resultante da divisão do monólito por módulos.
- **Arquitetura B:** Arquitetura resultante da divisão do monólito por *bounded contexts*.

A próxima figura representa visualmente a árvore hierárquica de decisão gerada:

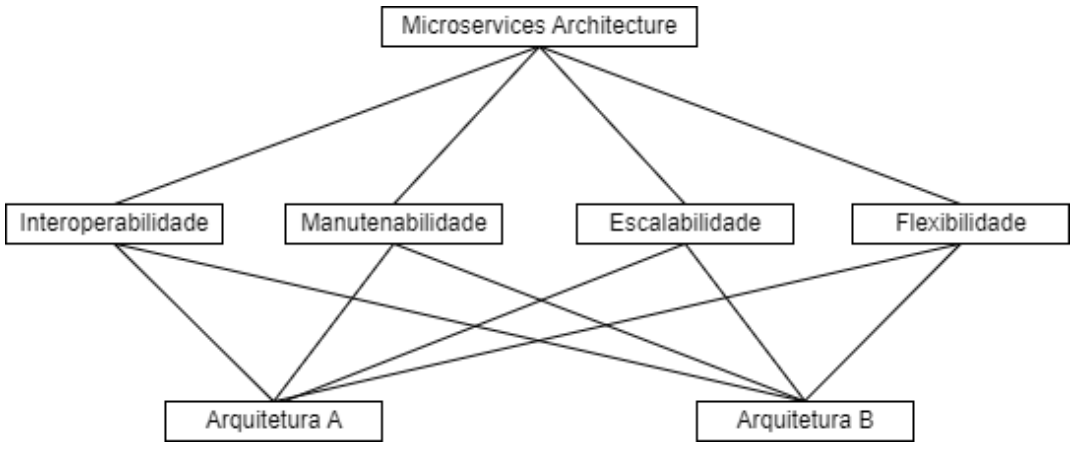


Figura 29 - AHP Árvore hierárquica de decisão

Tendo os critérios definidos e a árvore de hierarquias gerada, o próximo passo consiste na comparação da importância dos diferentes critérios. De forma a avaliar os critérios citados, é utilizada a seguinte escala de valores:

Nivel de importância	Definição	Explicação
1	Igual importânica	As duas atividades contribuem igualmente para o objetivo
3	Fraca importância	A experiência e o julgamento favorecem levemente uma atividade em relação à outra
5	Forte importância	A experiência e o julgamento favorecem fortemente uma atividade em relação à outra
7	Muito forte importância	Uma atividade é muito fortemente favorecida em relação a outra
9	Importância absoluta	A evidência favorece uma atividade em relação a outra com o mais alto grau de certeza
2,4,6,8	Valores intermediários	Quando se procura uma condição de compromisso entre duas definições

Figura 30 - Escala de valores de Satty

Tendo a escala definida, a seguinte tabela representa a matriz de comparação da importância de cada critério em relação aos restantes critérios:

Tabela 2 - Matriz de comparação de critérios

	Escalab…	**Manuten…**	**Flexibi…**	**Interoper…**
Escalabilidade	1	1/2	1/2	1/3
Manutenibilidade	2	1	1/2	1/3
Flexibilidade	2	2	1	1
Interoperabilidade	3	3	1	1

Após a determinação dos valores de importância, o próximo passo consiste em normalizar a matriz de comparação. A normalização da matriz é usada para determinar o vetor de prioridades, que é calculado através da média de valores da linha da tabela em questão (Nicola, n.d.-b).

Os valores de cada linha são calculados a partir da soma dos elementos de cada coluna, dividindo cada elemento da coluna pela soma calculada. A seguinte tabela representa a tabela de comparação de critérios normalizada:

Tabela 3 - Matriz de comparação de critérios normalizada

	Escalabilidade	**Manutenibilidade**	**Flexibilidade**	**Interoperabilidade**	**Vetor de Prioridades**
Esc...	1/8	1/13	1/6	1/8	0.12
Man...	2/8	2/13	1/6	1/8	0.17
Fle...	2/8	4/13	2/6	3/8	0.32
Int...	3/8	6/13	2/6	3/8	0.39

Analisando a normalização da tabela elaborada é possível compreender que o critério com mais relevância é a interoperabilidade, visto que um dos objetivos primários deste projeto passa por garantir que a plataforma antiga continua funcional durante o processo de migração. A escalabilidade, apesar da sua reconhecida importância, em comparação com os restantes critérios, acaba por ser a que possui um valor mais reduzido.

A próxima fase do método AHP têm como objetivo calcular a Razão de Consistência (RC) de forma a compreender e avaliar a consistência dos valores calculados na fase anterior. Se o RC calculado for superior a 0.1, os julgamentos não são confiáveis no sentido em que estão demasiados perto da aleatoriedade.

O RC é calculado com base no index de consistência (CI) (que é obtido com base λ_{max}) e no index de aleatoriedade (RI).

De forma a calcular-se λ_{max} deve-se, em primeiro lugar, multiplicar a matriz de comparação de critérios pelo vetor de prioridades gerado através da normalização dessa mesma matriz, como é possível verificar pelo seguinte cálculo:

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{2} & \frac{1}{3} \\ 2 & 1 & \frac{1}{2} & \frac{1}{3} \\ 2 & 2 & 1 & 1 \\ 3 & 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0.12 \\ 0.17 \\ 0.32 \\ 0.39 \end{bmatrix} = \begin{bmatrix} 0.49 \\ 0.69 \\ 1.29 \\ 1.58 \end{bmatrix}$$

Obtendo-se o a matriz resultante da multiplicação demonstrada, a próxima etapa consiste em dividir cada um dos elementos da matriz resultante pelo respetivo valor no vetor de prioridades, como é possível visualizar no seguinte cálculo:

$$\frac{\begin{bmatrix} 0.49 \\ 0.69 \\ 1.29 \\ 1.58 \end{bmatrix}}{\begin{bmatrix} 0.12 \\ 0.17 \\ 0.32 \\ 0.39 \end{bmatrix}} = \begin{bmatrix} 4.08 \\ 4.06 \\ 4.03 \\ 4.05 \end{bmatrix}$$

Após a obtenção da matriz resultante da divisão, resta agora calcular a média dos valores.

$$\lambda_{\max} = \frac{4.08 + 4.06 + 4.03 + 4.05}{4} = 4.055$$

CI pode então ser obtido através da seguinte fórmula, em que "n" representa o número de critérios utilizados.

$$CI = \frac{\lambda_{max} - n}{n - 1} = \frac{4.055 - 4}{4 - 1} = 0.018$$

Resta então determinar RI, que pode ser determinado com base na seguinte tabela, tendo que "n" corresponde à ordem da matriz.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.48	1.56	1.57	1.59

Figura 31 - Valores de Random Index (RI)

Tendo em conta que a ordem da matriz utilizada neste projeto é de ordem 4, assume-se 0.90 como valor de RI.

Tendo RI e CI sido determinados, tem-se todos os valores necessários ao cálculo de RC:

$$RC = \frac{CI}{RI} = \frac{0.018}{0.90} = 0.02$$

Visto que 0.02 é um valor inferior a 0.1, pode-se concluir que os valores das prioridades relativas estão consistentes.

A próxima etapa consiste na elaboração de matrizes individuais. Cada uma das matrizes deve representar a relevância que cada uma das alternativas dá a cada um dos critérios.

Desta forma, serão elaboradas matrizes e as respetivas normalizações para cada um dos critérios definidos. Os valores apresentados nestas matrizes foram discutidos e avaliados pela equipa de desenvolvimento.

As tabelas 4 e 5 representam a matriz e a sua normalização, respetivamente, referentes ao critério escalabilidade. O valor superior dado à Arquitetura B, neste critério, é referente à granularidade mais fina que esta apresenta.

Tabela 4 - Matriz de comparação de alternativas com base no critério de escalabilidade

	Arquitetura A	**Arquitetura B**
Arquitetura A	1	1/3
Arquitetura B	3	1

Tabela 5 - Matriz de comparação de alternativas com base no critério de escalabilidade normalizada

	Arquitetura A	**Arquitetura B**	**Vector de Prioridades**
Arquitetura A	1/4	1/4	0.25
Arquitetura B	3/4	3/4	0.75

As tabelas 6 e 7 representam a matriz e a sua normalização, respetivamente, referentes ao critério manutenibilidade. O valor superior dado à Arquitetura B, neste critério, é referente ao

alto nível de segregação e baixo nível de acoplamento que este apresenta, fatores estes que contribuem de forma decisiva para a manutenibilidade do sistema.

Tabela 6 - Matriz de comparação de alternativas com base no critério de manutenibilidade

	Arquitetura A	**Arquitetura B**
Arquitetura A	1	1/4
Arquitetura B	4	1

Tabela 7 - Matriz de comparação de alternativas com base no critério de manutenibilidade normalizada

	Arquitetura A	**Arquitetura B**	**Vetor de Prioridades**
Arquitetura A	1/5	1/5	0.20
Arquitetura B	4/5	4/5	0.80

As tabelas 8 e 9 representam a matriz e a sua normalização, respetivamente, referentes ao critério flexibilidade. O valor superior dado à Arquitetura B, neste critério, é referente ao alto nível de segregação e baixo nível de acoplamento que este apresenta, que facilita a adição de novas funcionalidades e alteração das existentes.

Tabela 8 - Matriz de comparação de alternativas com base no critério de flexibilidade

	Arquitetura A	**Arquitetura B**
Arquitetura A	1	1/2
Arquitetura B	2	1

Tabela 9 - Matriz de comparação de alternativas com base no critério de flexibilidade normalizada

	Arquitetura A	**Arquitetura B**	**Vetor de Prioridades**
Arquitetura A	1/3	1/3	0.33
Arquitetura B	2/3	2/3	0.66

As tabelas 10 e 11 representam a matriz e sua normalização, respetivamente, referente ao critério interoperabilidade. O valor superior dado à Arquitetura A, neste critério, é referente à quantidade inferior de modificações que esta arquitetura implica, em relação à Arquitetura B. Um dos objetivos principais deste projeto consiste em manter a plataforma antiga funcional durante a o processo de migração, fator este em que interoperabilidade influência de forma preponderante.

Tabela 10 - Matriz de comparação de alternativas com base no critério de interoperabilidade

	Arquitetura A	**Arquitetura B**
Arquitetura A	1	4
Arquitetura B	1/4	1

Tabela 11 - Matriz de comparação de alternativas com base no critério de interoperabilidade normalizada

	Arquitetura A	**Arquitetura B**	**Vetor de Prioridades**
Arquitetura A	4/5	4/5	0.80
Arquitetura B	1/5	1/5	0.20

Com base nos valores obtidos na coluna referente ao vetor de prioridades, torna-se possível escolher qual das alternativas propostas é a mais adequada à situação. A escolha é efetuada a partir do valor calculado através da multiplicação da matriz composta pelos vários vetores de prioridade resultantes das comparações entre alternativas pela matriz composta pelo vetor de prioridade resultante da comparação entre critérios.

Com este pressuposto, obtém-se o seguinte cálculo:

$$\begin{bmatrix} 0.25 & 0.20 & 0.33 & 0.80 \\ 0.75 & 0.80 & 0.66 & 0.20 \end{bmatrix} \begin{bmatrix} 0.12 \\ 0.17 \\ 0.32 \\ 0.39 \end{bmatrix} = \begin{bmatrix} 0.4816 \\ 0.5152 \end{bmatrix}$$

A matriz resultante da multiplicação mostra que o maior valor é 0.5152, valor esse correspondente à Arquitetura B. Pode-se assim afirmar que, com base no método AHP e os critérios definidos no mesmo, a Arquitetura B é a que melhor se enquadra no contexto atual do projeto.

4.4 Processo de desenvolvimento

O desenvolvimento de novos serviços pressupõe a gestão de um conjunto processos. Estes processos vão desde as boas práticas de desenvolvimento até à *release*, que consiste no processo de implantar as novas funcionalidades no ambiente de produção.

O processo atual consiste em utilizar GitHub como repositório de código, Jenkins como ferramenta de *Continuos Integration* (CI) & *Continuos Delivery* (CD) e Jira como plataforma para gerir *backlog*.

No entanto, foi tomada e decisão que para os novos serviços resultantes da migração, a ferramenta Jenkins seria abandonada, passando a utilizar-se GitHub *actions* como ferramenta

de CI e ArgoCD como ferramenta de CD. Esta decisão foi tomada pelo chefe de engenharia, tendo sido totalmente alheia ao autor deste documento.

Sendo assim, GitHub continua como repositório *git*, GitHub *actions* serão utilizadas para automatização de processos de integração, ArgoCD como controlador de aplicações e Jira como como plataforma para gerir *backlog*.

São utilizados os relatórios automaticamente gerados pelo Jira, por sprint, de forma a criar estatísticas e perceber o progresso da equipa em termos de velocidade na entrega de *story points*.

O processo de desenvolvimento pode ser descrito nos seguintes passos:

1. Criação de uma nova *branch* local de desenvolvimento a partir de "master".
   Esta *branch* deve respeitar a convenção de nomenclatura *kebab case* e possuir o respetivo identificador da tarefa em questão, de forma à ferramenta Jira conseguir identificar e associar a *branch* de desenvolvimento à respetiva tarefa.
2. Desenvolvimento da tarefa em questão.
   Os *commits* efetuados na *branch* de desenvolvimento devem ser atómicos e com uma mensagem descritiva da alteração em questão.
3. *Sync* do *branch* local, de forma a atualizá-lo a torná-lo remoto.
4. Abrir *pull request* para master. Neste passo ocorre uma pipeline de validação da *branch* em questão. Esta pipeline assegura não só a qualidade do desenvolvimento através de uma análise estática realizada pelo SonarQube, como também assegura que o novo desenvolvimento não provocou alterações indesejadas, através de uma execução de todos os testes. Caso a análise estática do SonarQube acuse erros ou não haja uma taxa de 100% de testes sucedidos, torna-se impossível o *merge* para "master".
5. *Code review* realizado por, pelo menos, 2 membros da equipa.
6. *Deploy* para ambiente de *staging*. É nesta fase que o desenvolvedor pode realizar os testes em um ambiente funcional e garantir que o comportamento da nova funcionalidade é o desejado.
7. *Deploy* para ambiente de UAT (*User Acceptance Testing*). É neste ambiente que o *product owner* valida o desenvolvimento efetuado.
8. *Merge* para "master".
9. *Deploy* para produção.

Como é possível verificar pelos passos supracitados, o desenvolvimento de uma funcionalidade leva a inúmeras etapas, sendo que muitas delas estão alheias ao desenvolvedor, como por exemplo o processo de *code review* e o processo de validação por parte do *product owner*.

Tendo em conta que, pela *definition of done* definida pela equipa, uma funcionalidade só é dada como concluída quando está *deployed* em produção, estes processos manuais acabam por prejudicar negativamente as estatísticas geradas.

Apesar de existir, no momento, uma pipeline de validação que facilita o processo de CI, não é possível de todo substituir as etapas manuais já citadas, visto que são cruciais para garantir a qualidade nas funcionalidades entregues.

4.5 Requisitos

Um requisito corresponde a uma necessidade ou expectativa que o sistema deve satisfazer. Estes servem como base para a conceção e desenvolvimento de um sistema e descrevem o que este deve fazer, assim como definir o seu comportamento e um conjunto de outras características.

Os seguintes tópicos descrevem os dois diferentes tipos de requisitos e enumeram os que foram identificados. A análise associada ao desenvolvimento destes mesmos requisitos é efetuada no capítulo de desenvolvimento.

4.5.1 *Requisitos funcionais*

Os requisitos funcionais são aqueles que definem as funcionalidades que um sistema deve possuir. Estes especificam as características do mesmo e têm como objetivo dar valor ao sistema através do solucionamento e/ou alívio de problemas dos clientes-alvo.

Os requisitos funcionais não foram identificados pelo autor deste documento e estão diretamente ligados ao plano definido pela empresa. Todos estes requisitos estão associados à segregação do módulo de OMS abordado em secções anteriores, em específico ao *bounded context* cujo objeto central denomina-se "PurchaseOrder". Com a conclusão destes requisitos torna-se possível o *rollout* do serviço desenvolvido.

Os requisitos funcionais identificados são:

- Desenvolvimento de um novo serviço de *PurchaseOrders*. Este serviço deve suportar:
    - **RF#1**: Fornecimento da informação relativa a todas *PurchaseOrders* associadas a uma marca.
    - **RF#2**: Importe de entidades via *spreadsheet*.
    - **RF#3**: Emissão de eventos associados a mudanças de estado da *PurchaseOrder*.
    - **RF#4:** Integração com o módulo 3PL de forma a poder comunicar com os sistemas dos armazéns.
    - **RF#5**: Integração com o módulo de 3PL de forma a poder receber informação aquando da mudança de estado de uma *PurchaseOrder*.

4.5.2 *Requisitos não funcionais*

Os requisitos não funcionais são aqueles que especificam restrições e/ou qualidades que um sistema deve possuir, mas que não estão diretamente ligadas a uma funcionalidade em específico. Estão normalmente ligados a questões de desempenho, segurança, manutenibilidade, escalabilidade etc.

Os requisitos não funcionais identificados são:

- **RNF#1**: Desenvolvimento de um projeto *template* que sirva de base a todos os serviços desenvolvidos. Este projeto base deverá definir a estrutura do projeto a seguir e definir estratégias de teste (unitários e integração) através da análise de bibliotecas a adotar.
- **RNF#2**: Seguir todas as restrições implementadas pela Maersk, incluindo alarmística, cobertura de testes (80%), qualidade de código e monitorização.
- **RNF#3**: Assegurar que todos os serviços permitem o funcionamento da plataforma Spoke enquanto a migração não estiver completa.
- **RNF#4**: O sistema deve garantir a melhor experiência possível ao utilizador, assegurando respostas rápidas através de, por exemplo, processamento assíncrono.

5 Desenvolvimento

Esta secção tem como objetivo demonstrar o trabalho realizado visando os objetivos proposto, entre as quais: desenvolvimento do projeto *template*, automatização de processos, e desenvolvimento da solução através da implementação dos requisitos funcionais e não funcionais identificados.

5.1 Projeto *template*

O projeto *template*, que servirá como por base a todos os serviços desenvolvidos, tem como principal objetivo a uniformização, através da definição de standards para padrões a adotar, bibliotecas a utilizar etc. Para além desta uniformização, o desenvolvimento deste projeto será importante para diminuir o tempo associado ao processo de criação de um novo serviço.

A análise e respetivo desenvolvimento deste projeto contribui para o alcançar do primeiro requisito não funcional identificado - **RNF#1**, correspondente ao objetivo **Uniformização de serviços.**

O projeto *template* será construído com base na arquitetura *onion*, já abordada neste documento, utilizando a última versão de .NET com *long term support (6.0).*

Após a instalação do *software development kit* (SDK) (*Install .NET on Windows - .NET | Microsoft Learn*, 2023) e, utilizando a .NET CLI (*..NET CLI | Microsoft Learn*, 2022), executou-se o comando: "*dotnet new webapi -n "dotnet-template""* de forma a criar o projeto de forma simples e com todas as dependências necessárias.

As seguintes secções pretendem demonstrar o trabalho realizado durante a fase de análise e desenvolvimento deste mesmo projeto.

5.1.1 *Arquitetura*

O primeiro passo após a criação do projeto passa por definir de forma clara a arquitetura a utilizar. Tendo por base os conceitos teóricos reunidos durante a fase de análise, a estrutura do projeto *template* pode ser vista da seguinte forma:

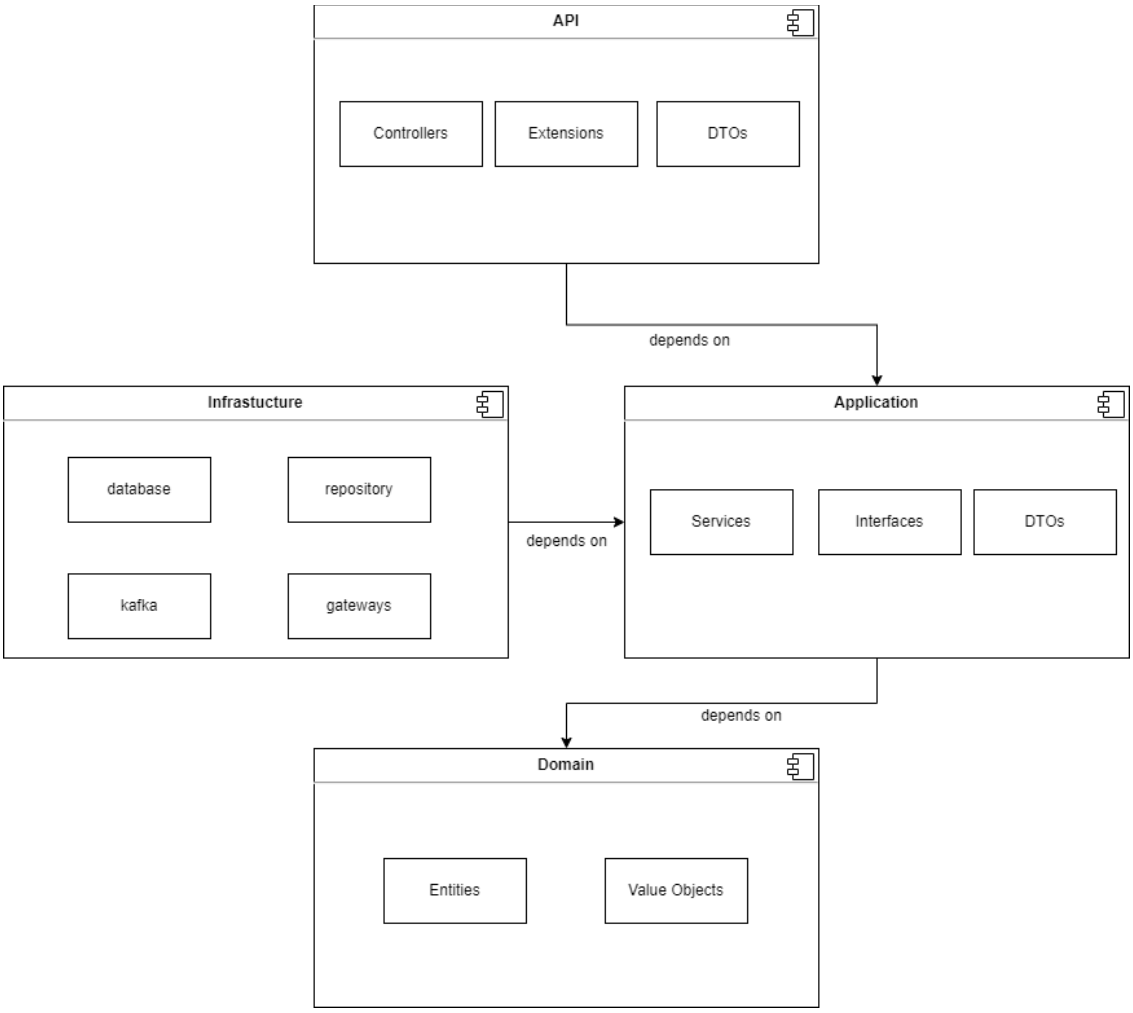


Figura 32 - Estrutura do projeto *template*

Com base neste diagrama é possível compreender a divisão por camadas e consequente subdivisão.

As diferentes camadas podem ser descritas da seguinte forma:

- **API**: A camada de API é a camada que expõe a aplicação para o exterior, através da abertura de *endpoints*, que neste diagrama são representados através de *controllers.* Para além disso, esta camada é o ponto de entrada da aplicação e deve ser responsável por instanciar todas as operações necessárias à execução da aplicação, como por exemplo injeção de dependências, conexão à base de dados, etc. Esta camada também

possui DTOs de forma a poder receber diferentes *inputs* e expor apenas a informação essencial à funcionalidade em questão.

- **Infrastructure**: A camada de infraestrutura é uma das camadas mais externas e é nesta camada que são realizadas todas as operações relativas a base de dados e comunicação com serviços externos. Nesta camada não deve ser implementada lógica de negócio nem qualquer tipo de caso de uso. Esta camada pode utilizar inúmeros protocolos de comunicação e implementar persistência de dados de forma a executar a funcionalidade esperada.
- **Application**: A camada de aplicação é a segunda camada mais interna desta arquitetura. É responsável por orquestrar os modelos de forma a executar funcionalidades específicas. Nesta camada são também definidas as regras da aplicação, que são regras utilizadas no contexto de uma funcionalidade específica. De forma a executar as funcionalidades, são utilizados:
    - **Services**: um bloco de código que implementa um caso de uso.
    - **Interfaces**: a camada de aplicação é responsável por coordenar operações que envolvem, por exemplo, escrita e leitura de uma base de dados. As funções a utilizar devem ser declaradas em interfaces na camada de aplicação e implementadas quer na camada de aplicação, quer na camada de infraestrutura.
    - **DTO**: um DTO é um objeto que contem informação que é passada entre diferentes camadas. É utilizado quando se deseja "transferir" informação que não está exposta no modelo de domínio.
- **Domain**: A camada de domínio é a cada mais interna desta arquitetura. É nesta camada que são definidas as entidades de domínio, que em conjunto com os *value objects,* representam o modelo de domínio da aplicação. É também nesta camada que são definidas as regras de negócio transversais a toda a aplicação.

5.1.2 *Desenvolvimento*

Tendo em conta o diagrama citado na secção anterior, que representa a estrutura do projeto *template,* e a respetiva descrição do mesmo, deu-se início ao desenvolvimento.

A primeira etapa do desenvolvimento passa por criar uma estrutura de diretórios clara para que se possa dar início ao processo de desenvolvimento.

A próxima imagem representa a estrutura de pastas do projeto:

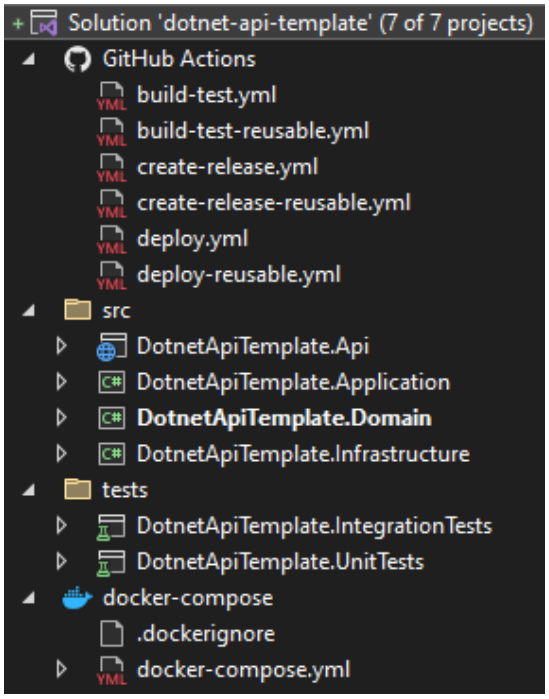


Figura 33 - Estrutura de pastas projeto *template*

A estrutura de diretórios pode ser descrita da seguinte forma:

- **GitHub Actions**: diretório responsável por armazenar as pipelines essenciais ao processo de CI/CD. Este projeto *template* disponibiliza 3 *actions* distintas: *build-test, create-release* e *deploy*.
- **Source (src)**: diretório responsável por armazenar toda a lógica associada ao funcionamento da aplicação.
- **Testes (tests)**: diretório responsável por armazenar todos os testes, quer unitários, quer de integração.

Para além disso, é possível verificar a existência de um ficheiro *docker-compose,* essencial à conteinerização (*O Que é Conteinerização? – Explicação Sobre Conteinerização – AWS*, n.d.) da solução.

Dentro do diretório "src", é possível verificar uma divisão clara daquelas que são as diferentes camadas impostas pela arquitetura *onion*.

As seguintes subsecções pretendem explicar detalhadamente cada uma das camadas que constituem a estrutura de diretórios apresentada.

5.1.2.1 API

A camada de *API* é o ponto de entrada do projeto, sendo ela responsável pela inicialização do projeto, incluindo todas as suas dependências, e exposição do mesmo através da abertura de *endpoints*.

A seguinte imagem representa a estrutura desta camada:

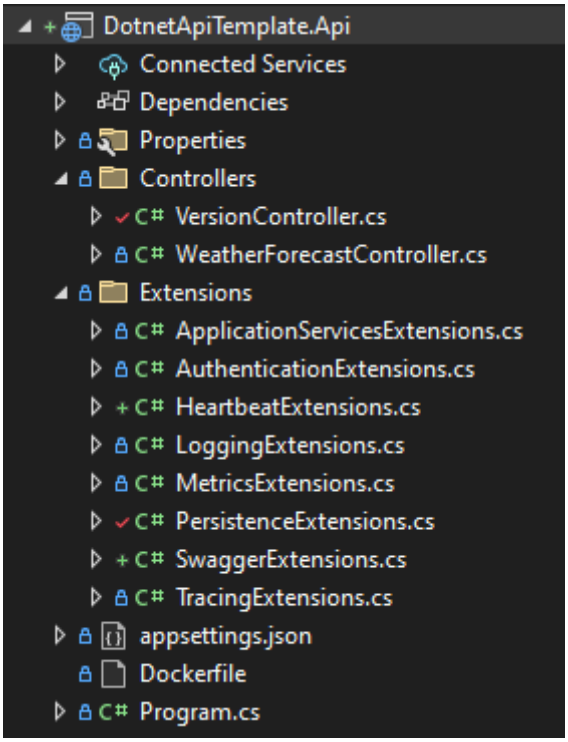


Figura 34 - Projeto *template* - camada API

O diretório "controller" será aquele responsável por armazenar pela exposição dos diversos *endpoints*, sendo que, como exemplo base, temos a exposição do *endpoint* de versionamento, que disponibiliza informação relativa à versão *deployed*:

```
[ApiController]
[Authorize]
[Route("[controller]")]
public class VersionController : ControllerBase
{

    private readonly IVersionService _versionService;

    public VersionController(IVersionService versionService)
    {
        _versionService = versionService;
    }

    [HttpGet(Name = "GetVersion")]
    public string GetVersion()
    {
        return _versionService.GetVersion();
    }
}
```

Figura 35 - *Version controller* e *endpoint*

Este *controller*, não apenas serve para expor informação relativa à versão atual do produto, mas também serve de exemplo àquele que algumas das práticas a adotar no desenvolvimento da aplicação: todas as dependências são declaradas como privadas e injetadas via construtor, onde a convenção de nomes deverá respeitar este mesmo exemplo.

Já o diretório *extensions* é responsável por armazenar todas as dependências do projeto, entre as quais autenticação, *logging,* base de dados, *swagger* e injeção de dependências.

Para autenticação será utilizado JWT (*JSON Web Tokens - Jwt.Io*, n.d.). A utilização de JWT como *token* utilizado para autenticação e autorização deve-se ao facto de este ser o mecanismo utilizado pela antiga plataforma.

A seguinte imagem representa a *extension* responsável pela configuração do *authentication scheme*:

```
public static class AuthenticationExtensions
{
    public static IServiceCollection AddApiAuthentication(
        this IServiceCollection services, ConfigurationManager configuration)
    {
        services
        .AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
        .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme,
                      options =>
                      {
                          configuration.Bind("JwtSettings", options);
                      });

        return services;
    }
}
```

Figura 36 - Authentication Extensions

A *keyword* "JwtSettings" visualizada nesta imagem é referente a um conjunto de configurações definidas no ficheiro "*appsettings.json"*, ficheiro este responsável por armazenar todas as configurações essenciais à aplicação.

Como configurações de base de dados: será utilizado a *entity framework como* ORM (*object-relaton mapper*) e PostgreSQL como RDBMS (*relational database management system*), onde a sua integração com .NET está disponibilizado através do *package* oficial da *entity framework* (*Npgsql Entity Framework Core Provider | Npgsql Documentation*, n.d.).

A escolha de *entity framework* teve por base o facto de esta ser a biblioteca oficial desenvolvida pela Microsoft, tendo como vantagem o facto de esta possuir *updates* constantes. Como alternativa foi analisado a biblioteca *Dapper* (*Welcome To Learn Dapper ORM - A Dapper Tutorial for C# and .NET Core*, n.d.), que é um micro ORM criado pela comunidade de desenvolvimento.

A seguinte imagem representa a *extension* responsável pela inicialização da camada de persistência, incluindo a inicialização da conexão à base de dados:

```
public static class PersistenceExtensions
{
    public static IServiceCollection AddPersistenceServicesAndRepositories(
        this IServiceCollection services,
        IConfiguration configuration)
    {
        services.AddDbContext<WeatherContext>(o =>
        o.UseNpgsql(
            configuration.GetConnectionString("dotnet-api-template"),
            options => options.EnableRetryOnFailure()));

        // Add persisence repositories
        services.AddSingleton<IWeatherForecastRepository, WeatherForecastRepository>();

        return services;
    }
}
```

Figura 37 - *Persistence Extensions*

5.1.2.2 Application

A camada da aplicação é onde se utiliza o domínio da aplicação de forma a executar funcionalidades específicas.

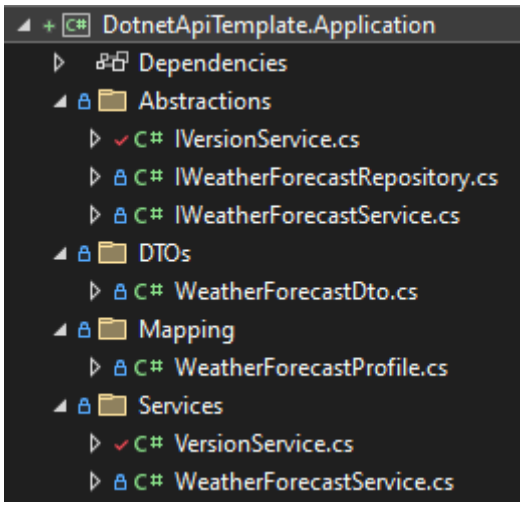


Figura 38 - Projeto *template* - camada de aplicação

O diretório de abstrações é responsável por armazenar as interfaces onde estão expostas as assinaturas dos métodos a utilizar. Estas abstrações são utilizadas, quer pela camada da API, de forma a utilizar serviços da camada de aplicação, quer pela camada de aplicação para utilizar lógica da camada de infraestrutura.

O diretório de "DTOs" é responsável por armazenar as classes responsáveis por expor apenas a informação necessária para a camada da API. O diretório de "*mapping*" é responsável pelas classes onde se encontra a lógica de mapeamento entre objetos de domínio e DTOs.

Por fim, a camada de serviços, onde são utilizadas as abstrações, os DTOs e os *mappers* de forma a executar funcionalidades específicas.

5.1.2.3 Domain

A camada de domínio é onde é declarado o domínio da aplicação em questão.

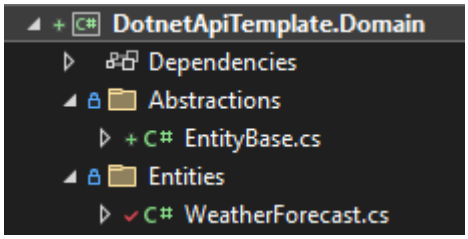


Figura 39 - Projeto *template* – camada de domínio

Esta camada conta, por norma, com pelo menos 2 diretórios distintos: "*abstractions*" onde é declarado as entidades abstratas, que são herdadas pelas entidades de domínio e "*entities*", onde são declaradas as entidades pelas quais o serviço está responsável de controlar.

5.1.2.4 Infrastructure

A camada de infraestrutura é onde é declarado configurações como base de dados e comunicações com serviços externos.

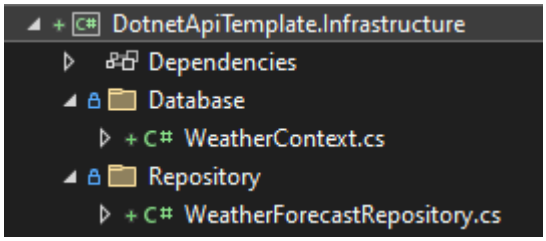


Figura 40 - Projeto *template* - camada de infraestrutura

O diretório "database" é responsável pelo armazenamento do contexto de base dados, onde são declaradas quais e de que forma as entidades de domínio são mapeadas para a base de dados.

O diretório "repository" é responsável por armazenar métodos de execução de chamadas à base de dados.

5.1.2.5 Tests

O diretório de testes está subdividido em 2 projetos de testes distintos: um direcionado a testes unitários e outro direcionado a testes de integração.

A prática de desenvolvimento de testes é algo imprescindível naquele que são os novos padrões de desenvolvimento impostos pela Maersk, sendo que todos os projetos são obrigados a uma cobertura de testes superior a 80% e a uma percentagem de duplicação de código inferior a 5%.

O estudo de diferentes bibliotecas para o desenvolvimento de testes foi essencial para a uniformização de práticas a utilizar entre as diferentes equipas.

A nível de testes unitários, o estudo de bibliotecas resultou na adoção das seguintes:

- **xUnit**: xUnit (*Unit Testing C# Code in .NET Core Using Dotnet Test and XUnit - .NET | Microsoft Learn*, n.d.) é uma *framework* de desenvolvimento de testes em .NET, sendo a *framework* mais usada a nível global para o desenvolvimento dos mesmos. Como alternativa, foi analisada também a framework NUnit (*Unit Testing C# with NUnit and .NET Core (Testes de Unidades Em C# Com NUnit e .NET Core) - .NET | Microsoft Learn*, n.d.), que foi a primeira *framework* de testes desenvolvida para .NET. A escolha de xUnit teve por base o facto de esta ser uma *framework* mais recente, com uma maior comunidade e cujo desenvolvimento teve em conta práticas de desenvolvimento que se pretende utilizar neste projeto, como por exemplo, TDD.
- **Moq**: Moq (*Unit Testing: Moq Framework | Microsoft Learn*, n.d.) é uma biblioteca que facilita a criação de *mock objects* que simulam o comportamento de objetos reais, de forma a poder testar componentes de forma isolada. Como alternativa, foi analisada também a biblioteca *NSubstitute* (*Testing in .NET Core with XUnit and NSubstitute - Level Up Coding*, n.d.). A escolha da biblioteca moq teve por base algumas das limitações conhecidas da biblioteca *NSubstitute*, como a incapacidade de testar métodos cujo modificador de acesso seja "*protected*".
- **AutoFixture**: AutoFixture (*UnitTest With AutoFixture In .NET 6.0*, 2022) é uma biblioteca que facilita o processo de criação de dados de teste. É essencial para otimizar o processo de criação de testes visto que a criação de *factories* para a criação de objetos é desnecessária. Como alternativa, foi analisada também a biblioteca *Bogus* (*Realistic Data Generation in .NET With Bogus - Code Maze*, n.d.), cujo propósito é exatamente o mesmo. A escolha da biblioteca AutoFixture teve por base a sua facilidade de implementação, apesar de ser reconhecido que a biblioteca *Bogus* têm algumas vantagens, como por exemplo, uma melhor performance na criação de grandes quantidades de informação.
- **FluentAssertions**: FluentAssertions (*Object Graph Comparison - Fluent Assertions*, n.d.) é uma biblioteca que providencia um conjunto extenso de métodos que permitem, de uma forma mais natural, comparar *outputs* com os resultados esperados. Apesar da *framework xUnit* já providenciar alguns dos métodos que esta biblioteca providencia, a adoção desta biblioteca teve por base a sua facilidade de leitura, implementação e vasta quantidade de métodos disponibilizados.

Já a nível de testes de integração, para além da utilização da *framework* xUnit e da biblioteca FluentAssertions acima descritas, o estudo de bibliotecas resultou na adoção das seguintes:

- **Testcontainers**: Testcontainers (*Testcontainers for .NET*, n.d.) é uma biblioteca que oferece suporte a testes com instâncias descartáveis de Docker *containers*. É uma ferramenta importante para garantir testes de integração com elevado nível de

confiança, garantindo a possibilidade de criação de *containers* para bases de dados, *message queues, etc.*

- **WireMock**: WireMock(*WireMock - Flexible, Open Source API Mocking | WireMock*, n.d.) é um simulador de *API's*, permitindo a criação de *mock servers.* Através da simulação de *endpoints* externos permite testar a aplicação com respostas reais, sem dependências externas.

5.1.2.6 Automatização de processos

Como já referido, serão utilizadas *GitHub Actions* de forma a automatizar processos de integração, sendo que para o âmbito do projeto atual foram desenvolvidas três *pipelines* distintas. Estas *pipelines* estão desenvolvidas de forma a serem reutilizáveis, recebendo inúmeros *inputs* e acedendo aos *GitHub secrets* (*Encrypted Secrets - GitHub Docs*, n.d.) de forma a guardar informação considerada sensível.

- ***Build, test and code analysis:*** esta pipeline tem como objetivo validar os *pull requests* através de um conjunto de ações cujo obtivo passa por tentar garantir qualidade naquilo que é o código desenvolvido:

```yaml
jobs:
  build_test_scan:
    name: Run tests and scan
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - name: Setup .NET Core SDK 6.0.x
        uses: actions/setup-dotnet@v2
        with:
          dotnet-version: 6.0.x

      - name: Login to GitHub NuGet registry
        run: dotnet nuget add source --username USERNAME --password ${{ secrets.NUGET_REGISTRY_TOKEN }} \
            --store-password-in-clear-text --name github "https://nuget.pkg.github.com/Maersk-Global/index.json"

      - name: Restore dependencies
        run: dotnet restore ${{ inputs.solution_name }}

      - name: Begin sonar scanner
        run: |
          dotnet tool install --global dotnet-sonarscanner
          dotnet sonarscanner begin \
            /d:sonar.host.url="${{ secrets.MDN_SONARQUBE_HOST }}" \
            /k:"${{ github.event.repository.name }}" \
            /d:sonar.login="${{ secrets.MDN_SONARQUBE_TOKEN }}" \
            /d:sonar.cs.opencover.reportsPaths="coverage/**/*.xml" \
            /d:sonar.exclusions="**/Migrations/**, **/Program.cs"

      - name: Build
        run: dotnet build ${{ inputs.solution_name }} -c Release --no-restore --verbosity quiet

      - name: Test
        run: |
          dotnet test ${{ inputs.solution_name }} \
            --no-restore \
            --verbosity quiet \
            --collect:"XPlat Code Coverage" \
            --results-directory coverage \
            -- DataCollectionRunSettings.DataCollectors.DataCollector.Configuration.Format=opencover

      - name: End sonar scanner
        run: dotnet sonarscanner end /d:sonar.login="${{ secrets.MDN_SONARQUBE_TOKEN }}"
```

Figura 41 - Pipeline Build, Test and Code analysis

A pipeline é composta por sete passos. Primeiramente, é instalado a versão de .NET a utilizar, seguido do login no *GitHub NuGet registry*. Este login é necessário visto que alguns dos

packages utilizados neste projeto são privados. Dada a existência destes mesmos *packages,* é necessário também executar o comando "*dotnet restore*", visto que este é responsável por procurar por dependências e instalá-las caso seja necessário (*Dotnet Restore Command - .NET CLI | Microsoft Learn*, 2023).

O próximo passo tem como objetivo instalar a ferramenta "*sonar-scanner"*, responsável pela execução da análise estática ao código. A análise começa através do comando "*sonarscanner begin*", sendo que esta é exportada para um ficheiro e integrada nos comentários do *pull-request.*

Por fim, a execução do build e, em seguida, os testes, dando por assim encerrada a ação do "*sonnarscanner*".

Não surgiu a necessidade nem a possibilidade de paralelizar processos pois todas os passos são de curta e duração e existem dependências diretas entre os diferentes passos.

- ***Create release:*** esta pipeline tem como objetivo criar uma *release* versionada, ou seja, um artefato *deployable* identificável:

```yaml
jobs:
  merge_job:
    if: github.event.pull_request.merged == true
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository and submodules
        uses: actions/checkout@v2
        with:
          ref: main
          fetch-depth: 0

      - name: Calculate the next version
        uses: codacy/git-version@2.4.0
        id: gitversion
        with:
          release-branch: main
          prefix: v

      - name: Create release
        uses: actions/github-script@v6
        env:
          TAG_NAME: ${{ steps.gitversion.outputs.version }}
          BODY: ${{ github.event.pull_request.body }}
        with:
          script: |
            const { TAG_NAME, BODY } = process.env;
            github.rest.repos.createRelease({
              owner: context.repo.owner,
              repo: context.repo.repo,
              tag_name: `${TAG_NAME}`,
              name: `${TAG_NAME}`,
              body: `${BODY}`
            });
```

Figura 42 - Create release pipeline

Esta pipeline conta com 3 passos: o primeiro, *checkout* do repositório atual, na *main branch*. Em seguida, cálculo da versão seguinte com base no registo de versões do GitHub e, por fim, a criação da release em si.

- ***Deploy:*** a pipeline de *deploy* tem como objetivo colocar os artefatos gerados pelo processo de build em um repositório, para que possa ser utilizado. É importante referir que este processo de *deploy* não é responsável por sincronizar o *container* com a nova versão *deployed*, sendo este um processo manual feito na interface do *ArgoCD*. Devido à extensão desta *pipeline*, a próxima imagem apenas aborda o passo onde ocorre o *push* da imagem gerada para o repositório da Azure:

```
- name: Build, tag, and push image to Azure ACR
  run: |
    source .version
    docker build -t ${{ secrets.AZURE_ACR_ENDPOINT }}/${{ inputs.ecr_repository }}:${IMAGE_TAG:1} -f
      src/${{ inputs.source_project_name }}/Dockerfile
          --build-arg NUGET_REGISTRY_TOKEN=${{ secrets.NUGET_REGISTRY_TOKEN }} --build-arg Version=${IMAGE_TAG:1} --push .
```

Figura 43 - Deploy pipeline

Tendo as 3 pipelines bases construídas, todos os projetos poderão reutilizar toda esta lógica, contribuindo para a diminuição do tempo associado à criação de novos serviços e respetivas pipelines, tal como pode demonstrar a seguinte imagem:

```yaml
name: Build and Test

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  test:
    name: Run tests
    uses: Maersk-Global/dotnet-api-template/.github/workflows/build-test-reusable.yml
    with:
      solution_name: dotnet-api-template.sln
    secrets: inherit
```

Figura 44 - Simplified Pipeline

Como é possível observar, apenas é feita referência à pipeline existente no projeto *template*, passando os parâmetros desejados através da *keyword "with".*

Com o desenvolvimento destas 3 *pipelines*, contribui-se diretamente para o objetivo referente a **Melhorias no processo de desenvolvimento**.

5.2 Desenvolvimento da solução

Esta secção tem como objetivo demonstrar o processo de desenvolvimento do novo microsserviço corresponde ao *bounded context* cujo agregado é a entidade "*PurchaseOrder*", *bounded contexto* este identificado no decorrer do processo de segmentação do módulo OMS.

Esta entidade sofreu uma mudança naquilo que é a sua nomenclatura, passando a denominar-se de "*Advanced Shipment Notice*" (ASN). Esta mudança é uma consequência direta de algumas diferenças conceituais existentes entre a antiga plataforma e a nova ECL *plataform*. Esta mudança é apenas referente à nomenclatura, sendo que o seu conceito se mantém o mesmo no contexto de negócio – entidade que representa aquilo que será a entrada de uma determinada quantidade de produtos em um armazém específico.

5.2.1 *Modelo de domínio*

O seguinte drama pretende representar aquele que é o domínio do microsserviço a desenvolver, o "ecl-asn-service":

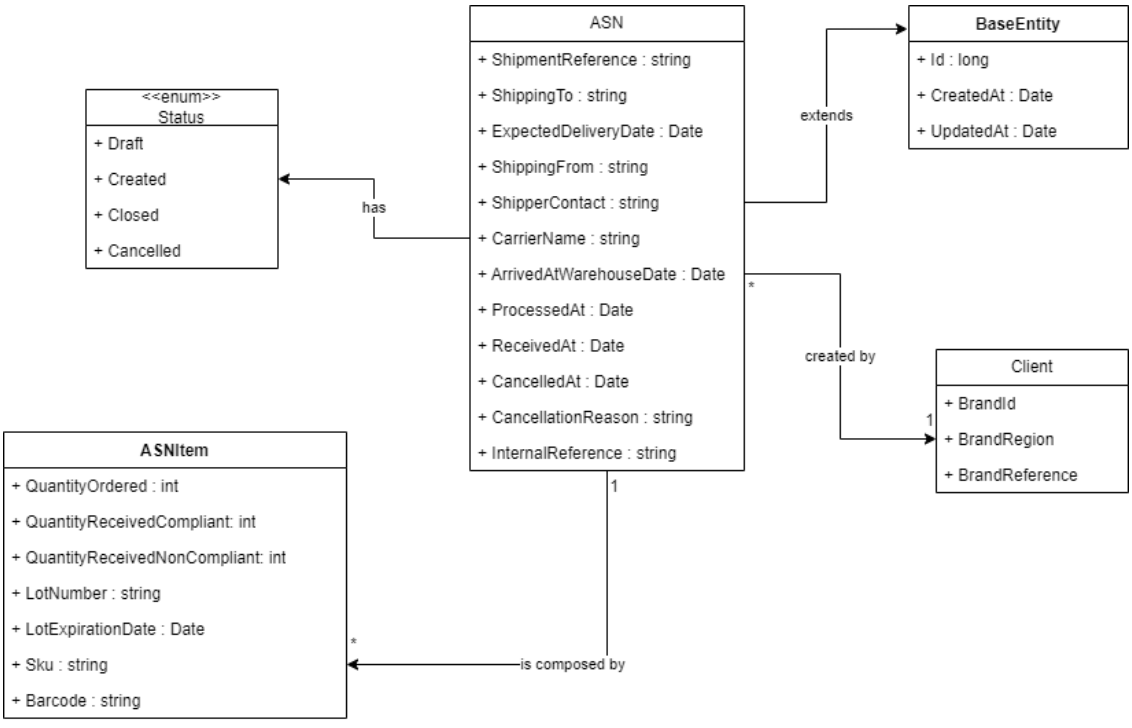


Figura 45 - ecl-asn-service modelo de domínio

5.2.2 *Desenvolvimento dos requisitos*

Esta secção visa expor o trabalho realizado para a concretização dos requisitos identificados anteriormente neste documento. O desenvolvimento destes requisitos permite o *rollout* do novo serviço pois, segundo os *product owners*, estão asseguradas as funcionalidades bases que tornam o processo viável.

O desenvolvimento, com sucesso, de todos os requisitos contribui para a completude do objetivo referente à "**Segmentação do monólito**", pois permite desanexar um conjunto de funcionalidades do monólito atual, passando-as diretamente para o novo serviço criado.

Os requisitos não funcionais **RNF#1** e **RNF#4** não são abordados de forma individual pois são abordados durante o desenvolvimento do projeto *template* e **RF#2,** respetivamente.

5.2.2.1 RF#1: Fornecimento de informação relativa a todas as ASNs de uma marca

Este requisito consiste na exposição de um *endpoint* GET cujo objetivo consiste em fornecer informação relativa a todas ASNs de uma marca. Este *endpoint* será utilizado por uma página *web*, que irá dispor a informação em uma tabela.

O *endpoint* deverá ser capaz de receber *query parameters* de forma a poder filtrar a informação de acordo com um conjunto predefinido de parâmetros. Para além destes parâmetros, este *endpoint* deverá suportar filtros de paginação. O seguinte diagrama de sequência pretende demonstrar a sequência de processos entre as diferentes camadas da aplicação:

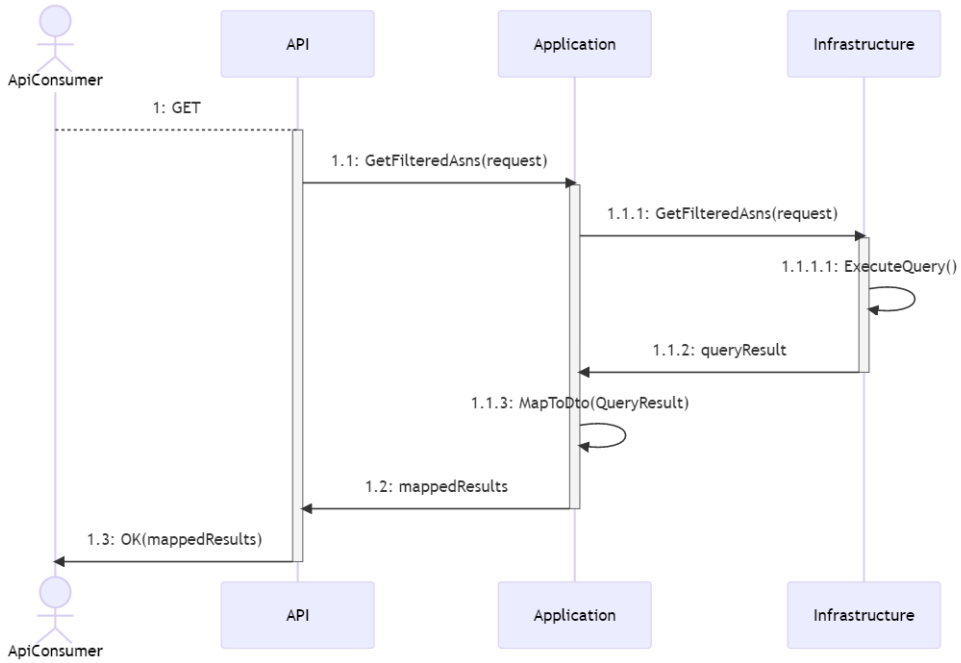


Figura 46 - RF#1: Diagrama de sequência

O primeiro passo para esta implementação passa pela criação de um *controller* para a entidade ASN. Este *controller* é responsável pela exposição dos *endpoints* e a definição das suas rotas:

```
[HttpGet]
[ProducesResponseType(StatusCodes.Status200OK, Type = typeof(PageResponse<AsnPagedInfoDto>))]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
2 references | - changes | -authors, -changes
public async Task<IActionResult> Get(
    [FromQuery] GetFilteredAsnRequest request,
    CancellationToken cancellationToken)
{
    if (!HttpContext.User.TryGetEmail(out _) ||
        !HttpContext.User.TryGetBrandReference(out var brandReference))
    {
        return Unauthorized();
    }

    var query = new GetFilteredAsnQuery(brandReference, request.PageSize, request.PageNumber,
        request.OrderBy, request.Status, request.ExpectedDeliveryDateLowerLimit,
        request.ExpectedDeliveryDateUpperLimit, request.ShippingTo, request.Search);

    var result = await _mediator.Send(query, cancellationToken);

    return Ok(result);
}
```

Figura 47 - RF#1: Controller

Com base nesta imagem é possível compreender que a referência da marca (*brandReference*) é extraído a partir de um token de autenticação. A implementação de toda a configuração associada a autenticação e autorização está fora do contexto deste documento e, sendo assim, não será abordada. Para além disso, é possível verificar que são aceites múltiplos *query parameters* (*Required Query String Parameters in ASP.NET Core - Code Maze*, n.d.) de forma a suportar, não só os filtros definidos, mas também os filtros de paginação.

Visando a segregação de dependências entre as camadas de *API* e *Application*, foi adotado um *design pattern* denominado *Mediator* (*Mediator*, n.d.). Este padrão permite que exista um mediador entre as diferentes dependências, sendo que este é o único responsável por redirecionar as chamadas para os componentes desejados.

A biblioteca adotada de forma a implementar este padrão denomina-se *MediatR* (*Mediator Pattern Com MediatR No ASP.NET Core | Blog TreinaWeb*, 2020), que é uma biblioteca extensa com uma enorme comunidade. Como alternativa, foi também analisada a biblioteca *FluentMediator* (Paulovich, 2019), cuja principal vantagem é a simplicidade de implementação.

A escolha da biblioteca levou principalmente em conta o facto de a primeira encontrar-se em permanente manutenção, enquanto a última não sofre qualquer *update* desde 2019 (Paulovich, 2019).

A implementação do componente responsável pelo *handle* deste pedido encontra-se na camada de aplicação:

```
public async Task<AsnPageDto> Handle(GetFilteredAsnQuery request, CancellationToken cancellationToken)
{
    var (rows, totalCount) = await _asnRepository.QueryWithFiltersAsync(request, cancellationToken);

    return new AsnPageDto
    {
        Results = rows.Select(x => x.MapToPagedDTO()),
        TotalResultsCount = totalCount,
        TotalPages = (int)Math.Ceiling((decimal)totalCount / request.PageSize),
        NextPageNumber = request.PageNumber * request.PageSize >= totalCount ? null : request.PageNumber + 1
    };
}
```

Figura 48 - RF#1: Application

Como última dependência existe a camada de infraestrutura, responsável pelo armazenamento dos repositórios para a entidade em questão.

Nestes repositórios encontram-se declaradas as *queries*, construídas com base na integração biblioteca *LINQ* com a *entity framework* (Wickramarathna, 2021).

A seguinte imagem representa a query resultante dessa mesma integração, aplicando também os filtros, ordenação e paginação:

```
public async Task<(IReadOnlyList<Domain.Asn.Asn>, int totalRows)> QueryWithFiltersAsync(
    GetFilteredAsnQuery queryRequest, CancellationToken cancellationToken)
{
    var query = _dbSet
        .AsNoTracking()
        .Where(x => x.Client.BrandReference == queryRequest.BrandReference);

    query = ApplyFilters(queryRequest, query);

    query = ApplyOrderBy(queryRequest.OrderBy, query);

    var totalRows = await query.CountAsync(cancellationToken);

    var listResult = await query
        .Include(x => x.Items)
        .Skip(queryRequest.PageSize * (queryRequest.PageNumber - 1))
        .Take(queryRequest.PageSize)
        .ToListAsync(cancellationToken);

    return (listResult, totalRows);
}
```

Figura 49 - RF#1: Infrastructure

A seguinte imagem demonstra aquilo que é uma resposta tipo para um pedido efetuado a este *endpoint*, utilizando os filtros de paginação de forma a obter um *sample* simples e preciso, ou seja, apenas uma ASN:

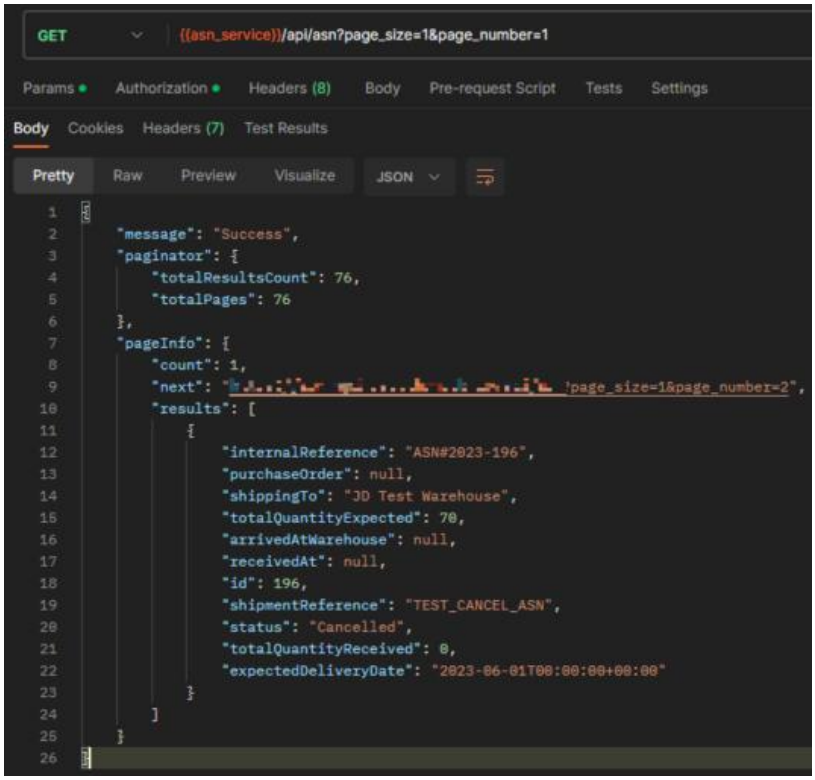


Figura 50 - RF#1: resposta do endpoint

5.2.2.2 RF#2: Importe de entidades via *spreadsheet*

Como é possível visualizar no modelo de domínio deste serviço, uma ASN pode ser composta por um ou mais *ASNItems*. Cada um destes *items* corresponde a um determinado produto que um dado armazém irá receber, em quantidades específicas.

A equipa responsável pelo produto em desenvolvimento tomou a decisão de que a página *web* responsável pela criação de ASNs deveria suportar o *upload* de uma *spreadsheet*, *spreadsheet* esta que contêm toda a informação relativa aos *items* em questão e que a mesma deve ser validada e processada neste serviço. Esta decisão foi completamente alheia ao autor deste documento. Para além do *upload* de uma *spreedsheet*, essa mesma página *web* requer o preenchimento de um formulário corresponde às informações essenciais à criação de uma ASN.

Esta requisito funcional levou a uma reflexão com o objetivo de compreender como garantir tempos de resposta rápidos tendo em conta que a validação do *spreedsheet* pode ser um processo demorado.

O seguinte diagrama de sequência pretende demonstrar o fluxo de sucesso associado à execução do *endpoint* de criação da entidade ASN:

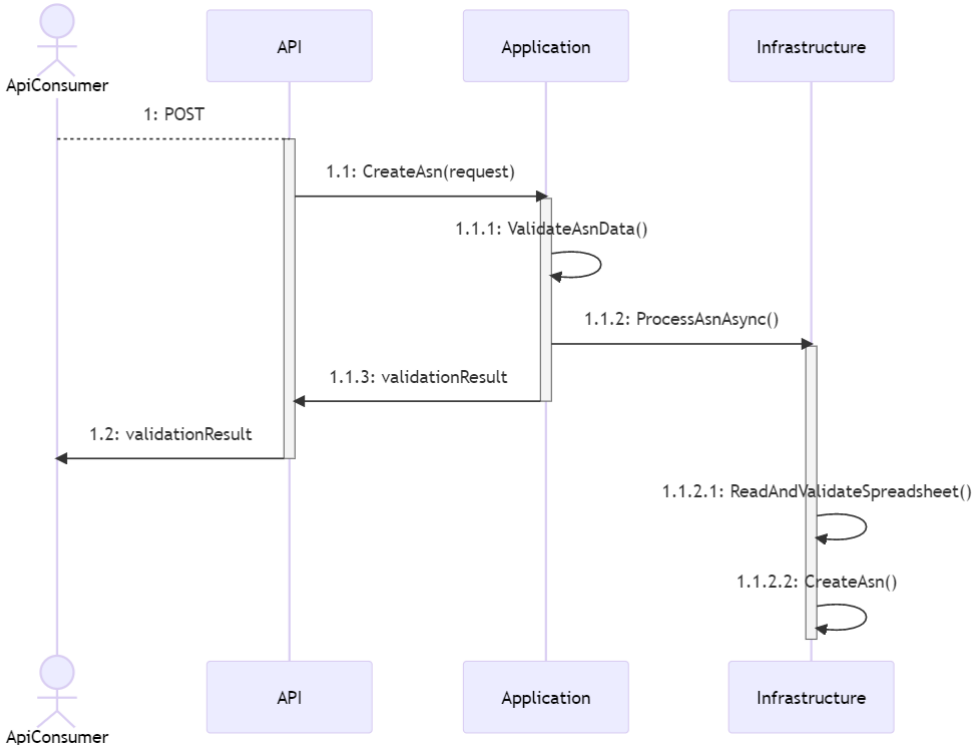


Figura 51 - RF#2: Diagrama de sequência

A primeira etapa da implementação deste requisito consiste na abertura de um novo *endpoint* que permita a criação de uma ASN. Este *endpoint* está exposto no mesmo *controller* referido no **RF#1.**

A sua implementação é em todo semelhante ao requisito anterior, utilizando o padrão *mediator* para executar a funcionalidade que se encontra implementada na camada de aplicação:

```
[HttpPost("ImportAsn")]
[ProducesResponseType(StatusCodes.Status202Accepted)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
3 references | Ademar Vale, 74 days ago | 3 authors, 5 changes | 5 work items
public async Task<IActionResult> Post([FromBody] AsnInformation asnInfo, IFormFile file)
{
    if (!HttpContext.User.TryGetEmail(out var tokenEmail) ||
        !HttpContext.User.TryGetBrandReference(out var brandReference))
    {
        return Unauthorized();
    }

    HttpContext.User.TryGetRegion(out var brand_region);
    asnInfo.BrandReference = brandReference;

    using var fileStream = file.OpenReadStream();

    var command = new ImportAsnCommand(asnInfo, file.FileName, fileStream, file.ContentType, tokenEmail, brand_region!);
    var result = await _mediator.Send(command);

    if (!result.IsValid)
    {
        return BadRequest(string.Join("\n", result.Errors));
    }

    return Accepted();
}
```

Figura 52 - RF#2: Controller

Como é possível verificar, este *endpoint* espera receber, para além de um conjunto de informações no body do pedido, um ficheiro.

A implementação do componente responsável pelo *handle* do pedido de criação encontra-se na camada de aplicação:

```
public async Task<ValidationResult> Handle(
    ImportAsnCommand request,
    CancellationToken cancellationToken)
{
    if (await ShipmentReferenceAlreadyExistsAsync(
        request.AsnInfo.ShipmentReference!,
        request.AsnInfo.BrandReference!,
        cancellationToken))
    {
        return new ValidationResult(new[]
        {
            new ValidationFailure("ShipmentReference", "Shipment already exists!")
        });
    }

    var warehouses = await _warehouseRepository.FindAsync(
        new WarehouseByReferenceAndRegionSpec(
            request.AsnInfo.ShippingTo!,
            request.Region),
        cancellationToken);

    if (ValidateFileHeaders(request.FileContent, warehouses.Single().HasLotManagement) is not true)
    {
        return new ValidationResult(new[]
        {
            new ValidationFailure("Template", "Wrong Template")
        });
    }

    var fileName = $"{_dateTime.UtcNow:yyyyMMddHHmmssfff}-{request.Filename}";
    var importFileUrl = await _remoteFileStore.SaveAsync(fileName, request.FileContent, cancellationToken);

    // create import row in db
    var asnEntity = AddAsnImportInDatabase(importFileUrl, request);

    await _unitOfWork.SaveEntitiesAsync(cancellationToken);

    // Enqueue Hangfire task
    _jobScheduler.Enqueue<IAsnImportService>(x => x.ProcessAsync(asnEntity.Id, cancellationToken));

    return new ValidationResult();
}
```

Figura 53 - RF#2: Application

Esta funcionalidade, tal como é possível visualizar no diagrama de sequência, está dividida em 2 partes distintas:

1. **Processamento síncrono**: a entidade ASN possui como index único a combinação das propriedades “brandReference” e “shipmentReference”. Com base nesta informação, a primeira validação passa por verificar se a referência introduzida pelo utilizador é única. A segunda validação passa por garantir que a *spreadsheet* recebida via *API* contém o *template* esperado. Por fim, o *import* é guardado na base de dados e é utilizado a biblioteca *Hangfire* (*Hangfire – Background Jobs and Workers for .NET and .NET Core*, n.d.), para que a leitura e validação da *spreadsheet* ocorra de forma assíncrona. Esta biblioteca contém uma extensão de funcionalidade que permite a execução de *background jobs* de forma simples. Como alternativa, foi analisada também a biblioteca *Quartz* (*ASP.NET Core Integration | Quartz.NET*, 2023), que até conta com uma comunidade maior que a primeira biblioteca. A decisão tomada teve por base o facto de que a biblioteca *Hangfire* disponibiliza um *dashboard* por defeito,

dashboard este que permite controlar os *backgroud jobs* configurados e obter dados estatísticos como, por exemplo, tempo de processamento.

2. **Processamento assíncrono**: tendo em conta que o tempo associado à leitura e validação da *spreadsheet* varia com base no tamanho dela, foi tomada a decisão de garantir que este processamento ocorra assincronamente. Desta forma consegue-se tempos de resposta curtos ao cliente que consuma este *endpoint* e contribui-se também para o **RNF#4.** Este processamento assíncrono é responsável pela leitura e validação da *spredsheet* e lidar com os cenários de sucesso e insucesso. O caso de insucesso não entra no contexto desta história por decisão da equipa de produto. No ao cenário de sucesso ocorre o *commit* da nova ASN para a base de dados e é emitido um evento associado à sua criação. Esta última parte é abordada com mais pormenor no próximo requisito funcional.

5.2.2.3 RF#3: Emissão de eventos associados a mudanças de estado

Todos os estados da entidade ASN correspondem a um momento distinto naquela que é o processo de recebimento de produtos no armazém:

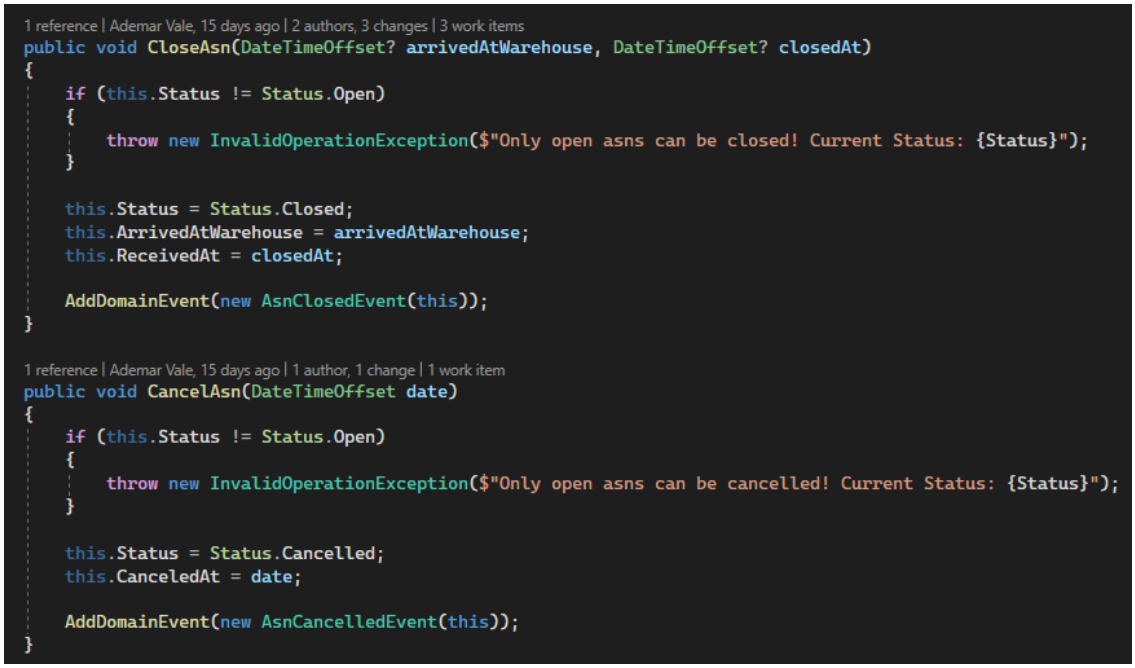
1. **Created**: Este estado corresponde à criação de uma ASN. É um processo normalmente iniciado pela marca, que pretende que um novo lote de produtos chegue a um determinado armazém. Aquando desta criação, o armazém é também notificado, via *API*, que determinadas quantidades de determinados produtos são esperadas, acompanhados pela data expectável dessa mesma chegada.
2. **Closed**: Este estado corresponde ao recebimento dos produtos correspondentes a uma ASN em um determinado armazém. De forma ao sistema ser notificado deste recebimento, é esperada uma comunicação via *API* proveniente do armazém. Estas comunicações têm o sentido exatamente oposto ao de criação.
3. **Cancelled**: Este estado corresponde ao cancelamento de uma ASN. Assim como estado "*closed*", esta comunicação é proveniente do armazém.

Estes estados representam diferentes momentos, porém fulcrais na gestão de armazém, como, por exemplo, a subida de *stock* em um determinado armazém sempre que uma ASN é *"closed".*

Com base neste mesmo pressuposto e, de forma a garantir que, futuramente, todos os serviços interessados encontram-se atualizados com as ocorrências deste serviço, tomou-se a decisão de garantir que um evento é produzido sempre que ocorre uma mudança de estado.

Para esta implementação adotou-se o padrão de *domain events* (*Domain Events: Design and Implementation | Microsoft Learn*, n.d.).

Sendo assim, na camada de domínio foi desenvolvida lógica que suporte este mesmo padrão:



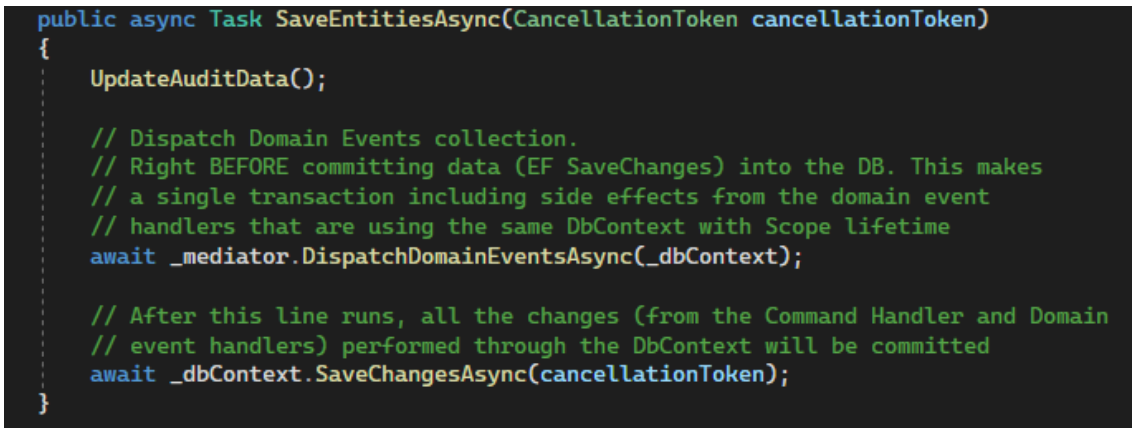
```
1 reference | Ademar Vale, 15 days ago | 2 authors, 3 changes | 3 work items
public void CloseAsn(DateTimeOffset? arrivedAtWarehouse, DateTimeOffset? closedAt)
{
    if (this.Status != Status.Open)
    {
        throw new InvalidOperationException($"Only open asns can be closed! Current Status: {Status}");
    }

    this.Status = Status.Closed;
    this.ArrivedAtWarehouse = arrivedAtWarehouse;
    this.ReceivedAt = closedAt;

    AddDomainEvent(new AsnClosedEvent(this));
}

1 reference | Ademar Vale, 15 days ago | 1 author, 1 change | 1 work item
public void CancelAsn(DateTimeOffset date)
{
    if (this.Status != Status.Open)
    {
        throw new InvalidOperationException($"Only open asns can be cancelled! Current Status: {Status}");
    }

    this.Status = Status.Cancelled;
    this.CanceledAt = date;

    AddDomainEvent(new AsnCancelledEvent(this));
}
```

Figura 54 - RF#3: Domain

Os eventos são produzidos para uma plataforma *Kafka,* desenvolvida e mantida internamente por uma equipa dedicada apenas a este mesmo propósito. A integração com esta plataforma é feita com a utilização de uma biblioteca também desenvolvida e mantida internamente.

Este requisito funcional exige a compreensão e implementação de inúmeros conceitos associados a *event streaming*, como por exemplo, *schema registry* (*Schema Registry Overview | Confluent Documentation*, n.d.), *topic partitions* (*Introduction to Apache Kafka Partitions*, n.d.), *etc.*

É importante referir que estes eventos apenas são despoletados quando as alterações são efetivamente *commited* para a base de dados, de forma a garantir consistência de dados dentro e fora do serviço:



```
public async Task SaveEntitiesAsync(CancellationToken cancellationToken)
{
    UpdateAuditData();

    // Dispatch Domain Events collection.
    // Right BEFORE committing data (EF SaveChanges) into the DB. This makes
    // a single transaction including side effects from the domain event
    // handlers that are using the same DbContext with Scope lifetime
    await _mediator.DispatchDomainEventsAsync(_dbContext);

    // After this line runs, all the changes (from the Command Handler and Domain
    // event handlers) performed through the DbContext will be committed
    await _dbContext.SaveChangesAsync(cancellationToken);
}
```

Figura 55 - RF#3: Infrastructure

5.2.2.4 RF#4: Integração com o módulo de 3PL de forma a poder comunicar com os armazéns

De acordo com a análise já efetuada ao sistema atual, é possível compreender que o antigo monólito utilizava um serviço denominado de 3PL (*third-party logistics*) de forma a estabelecer comunicação com os diversos sistemas de armazéns com os quais integra. Este serviço está construído para que a comunicação seja bidirecional: o nosso sistema é capaz de comunicar a criação de entidades para os sistemas dos armazéns e, por norma, os armazéns comunicam as atualizações dessas mesmas entidades.

Desta e forma e, de acordo com a gestão de estados já referenciados no **RF#3**, surgiu a necessidade de integrar o novo serviço com o serviço de 3PL. É importante referenciar que, assim como o antigo monólito, este módulo está desenvolvido em Python, fator este levado em conta e que exigiu cooperação com colaboradores experientes nessa mesma tecnologia.

Esta integração consistiu na produção e consumo do evento "**AsnCreatedEvent".** Este evento é despoletado no final do processamento assíncrono associado ao processo de criação de uma ASN, processo este abordado durante a análise e desenvolvimento do **RF#2**. O módulo de 3PL necessita consumir este evento de forma a poder comunicar a nova ASN aos armazéns integrados.

Esta integração pode ser visualizada no seguinte diagrama de sequência:

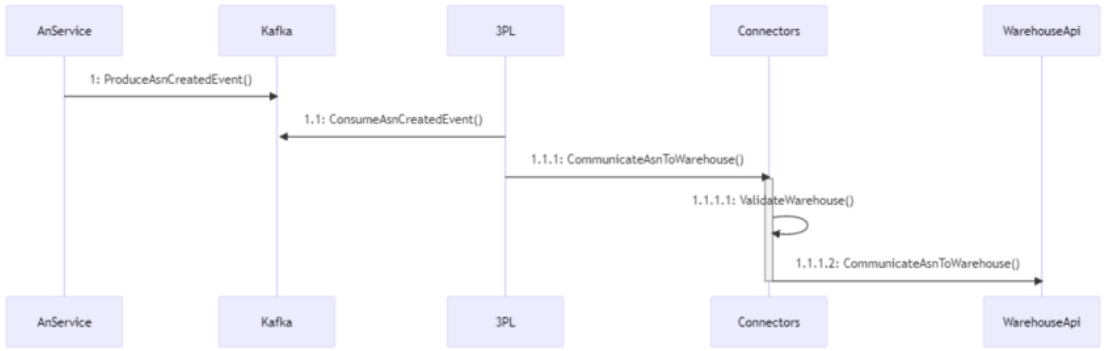


Figura 56 - RF#4: Diagrama de sequência

Sendo assim, procedeu-se à implementação de um novo *consumer* no serviço de 3PL que esteja subscrito ao tópico para o qual este evento é publicado:

```python
class Command(BaseCommand):
    must_exit = Event()

    def __init__(self):
        super().__init__()
        self.help = 'Launch a Retina consumer, for asn events'
        config = settings.RETINA["config"]
        config['enable.auto.offset.store'] = 'false'

        registry_config = copy.deepcopy(settings.SCHEMA_REGISTRY)["config"]
        poll_timeout = settings.RETINA["consumer_service"]["poll.timeout"]
        event_config = settings.RETINA["events"]["asn-created"]

        topics = [event_config["topic"]]

        self.handlers = {
            "AsnCreatedEvent": lambda sender, msg, _: signals.asn_created_event.send(
                sender=sender, evt=ASNCreated(**msg)
            ),
        }

        dispatcher = AvroDispatcher(self.handlers, logger)

        self.listener = KafkaAvroListener(
            kafka_config=config,
            registry_config=registry_config,
            evt_config=event_config,
            topics=topics,
            dispatcher=dispatcher,
            logger=logger,
            poll_timeout=poll_timeout,
        )
```

Figura 57 - RF#4: ASNCreatedEvent consumer

A lógica associada ao *handle* deste evento foi um reaproveitamento da lógica já implementada associada ao consumo do antigo evento associado à criação de uma “PurchaseOrder”, sendo apenas necessária a implementação do respetivo mapeamento de entidades:

```python
def handler(sender, evt: ASNCreated, **kwargs):
    brand_reference = Brand.objects.get(id=evt.client["brand_reference"]).reference
    event = {
        "timestamp": get_current_timestamp(),
        "shipping_to": evt.shipping_to,
        "brand_reference": brand_reference,
        "shipment_reference": evt.shipment_reference,
        "internal_reference": evt.internal_reference,
        "expected_delivery_date": evt.expected_delivery_date,
        "created_at": evt.created_at,
        "purchase_order": _purchase_order_payload(evt),
        "items": _items_payload(evt),
    }

    with ConfluentKafkaAvroProducer(kafka=settings.RETINA["config"],
                                    registry=settings.SCHEMA_REGISTRY["config"], logger=None) as producer:

        producer.produce(topic=settings.RETINA["events"]["ext-purchaseorder-created"]["topic"],
                         item_type_name="ExtPurchaseOrderEvent", msg=event,
                         on_delivery=producer_delivery_report)
```

Figura 58 - RF#4: AsnCreatedEvent handler

5.2.2.5 RF#5: Integração com o módulo de 3PL de forma a poder receber informação aquando da mudança de estado de uma *ASN*.

Como já referido, o processo de atualização de uma ASN têm origem em uma comunicação via API proveniente dos sistemas dos armazéns. Esta comunicação é feita via um *connectors* específico de cada sistema de armazém¸ que comunicam com 3PL via API. A comunicação entre o serviço de 3PL e o *ecl-asn-service* é efetuada via evento.

Esta integração consiste em dois fluxos distintos:

- **Fluxo de recebimento de uma ASN**: Para este fluxo foram desenvolvidos 2 eventos distintos - **CloseAsnCommand e AsnClosedEvent.** O seguinte diagrama de sequência ilustra o fluxo desta mesma funcionalidade:

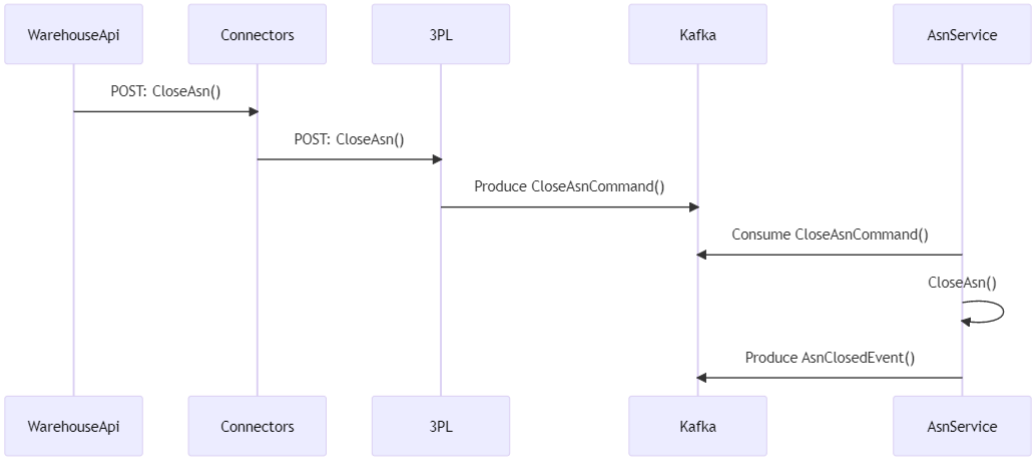


Figura 59 - RF#5: CloseAsnCommand e AsnClosedEvent

- **Fluxo de cancelamento de uma ASN:** Para este fluxo foram desenvolvidos 2 eventos distintos - **CancelAsnCommand e AsnCancelledEvent**. Do ponto de vista arquitetural e de implementação, é um fluxo em todo semelhante ao fluxo associado ao recebimento de uma ASN.

5.2.2.6 RNF#2: Seguir todas as restrições implementadas pela Maersk, incluindo alarmística, cobertura de testes (80%), qualidade de código e monitorização

O desenvolvimento deste requisito começou durante o desenvolvimento do projeto *template*, onde durante o desenvolvimento da secção relativa à automatização de processos, na pipeline relativa ao processo de *build-test*, se o utilizou o "*sonnar scanner"* de forma a recolher dados estatísticos resultantes do resultado da execução dos testes.

O resultado da análise é então publicada aquando da abertura de um *pull request,* tal como demonstra a seguinte imagem:

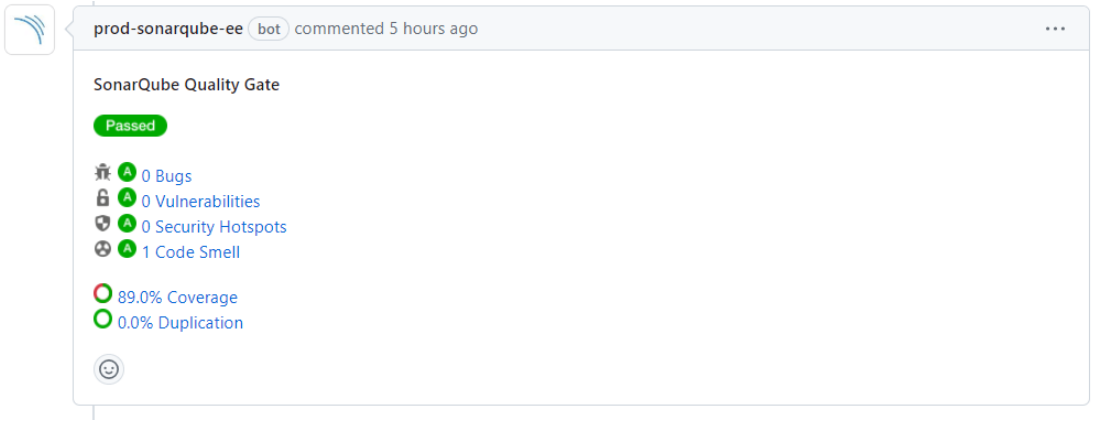


Figura 60 - RNF#2: Análise do Sonarqube

Com base no resultado da análise e, de acordo com as normas estipuladas pela Maersk, o *pull request* fica então bloqueado ou pronto para *merge*, tal como é possível visualizar na seguinte imagem, onde o resultado é negativo visto faltarem as aprovações necessárias*:*

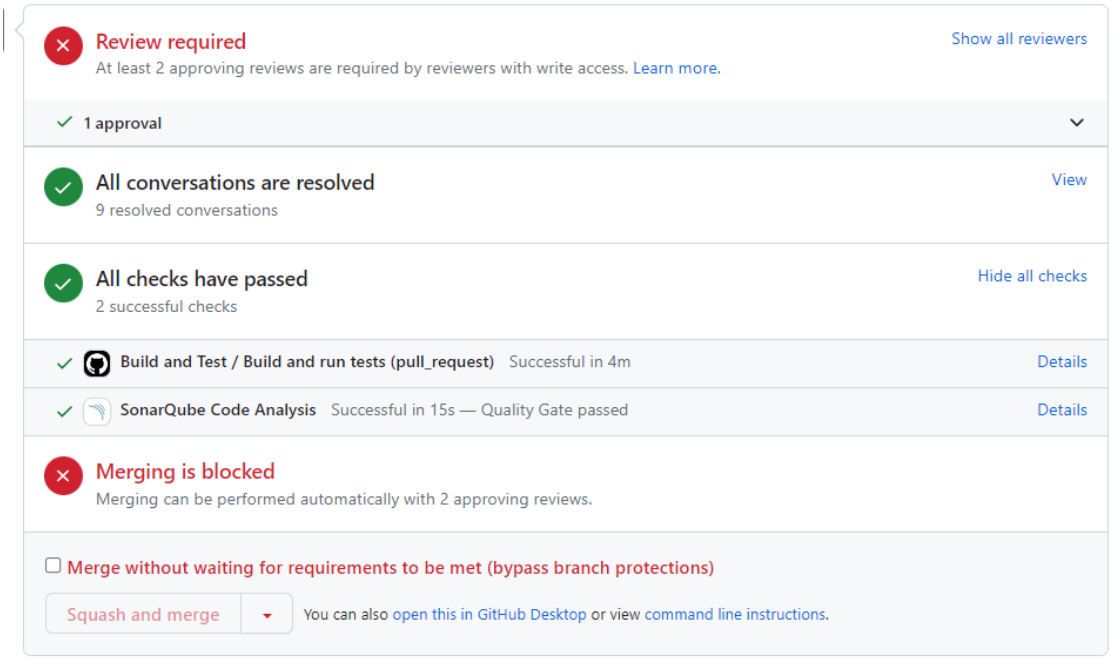


Figura 61 - RNF#2: Resultado da análise SonarQube

Relativamente à monitorização e alarmística, como já referido neste documento, irá ser utilizada a integração com o *sentry* visto esta ser uma ferramenta aprovada e utilizada pela Maersk em outros projetos.

De forma a implementar essa mesma integração, foi adotada a biblioteca *serilog*, pois para além de ser uma biblioteca mais utilizadas a nível global, é uma das bibliotecas recomendadas pela página oficial do *sentry* (*Serilog | Sentry Documentation*, n.d.).

A seguinte imagem demonstra a implementação associada à integração com o *serilog*:

```
public static class LoggingExtensions
{
    1 reference | ademar-vale-maersk, 203 days ago | 1 author, 1 change | 1 work item
    public static ConfigureHostBuilder AddApiLogging(
        this ConfigureHostBuilder hostBuilder, string serviceVersion)
    {
        hostBuilder.UseSerilog(
            (hostingContext, services, loggerConfiguration) => loggerConfiguration
                .ReadFrom.Configuration(hostingContext.Configuration)
                .ReadFrom.Services(services)
                .Enrich.FromLogContext()
                .WriteTo.Sentry(options =>
                {
                    options.InitializeSdk = true;
                    options.Dsn = hostingContext.Configuration["Sentry:Dsn"];
                    options.Environment = hostingContext.HostingEnvironment.EnvironmentName;
                    options.Release = serviceVersion;
                    options.ReportAssembliesMode = Sentry.ReportAssembliesMode.None;
                    options.AttachStacktrace = true;
                    options.StackTraceMode = Sentry.StackTraceMode.Original;
                }));

        return hostBuilder;
    }
}
```

Figura 62 - Integração com Sentry

Desta forma torna-se possível a visualização e monitorização de erros, pois para além de garantir que todas as *exceptions* são capturadas, torna-se possível acionar mecanismos de alarmística, como, por exemplo, notificações no Microsoft Teams ou via *e-mail*.

5.2.2.7 RNF#3: Assegurar que o novo microsserviço mantem o funcionamento da plataforma Spoke enquanto a migração não estiver completa

Idealizando um muito perfeito, a nova plataforma *web* estaria pronta e todas as equipas terminariam o processo de migração em tempos semelhantes, o que levaria à não necessidade de manter a antiga plataforma ativa, passando-se a usar apenas a nova plataforma. A verdade é que não se pode esperar por um cenário destes e, é necessário ter em consideração que múltiplos processos e páginas *web* ainda utilizam a base de dados do monólito como fonte da verdade.

Este facto é ainda mais preponderante quando, durante o desenvolvimento, percebeu-se que algumas das páginas *web* que ainda se encontram disponíveis aos utilizadores foram construídas com base numa *framework* denominada *web2py* (*What Is Web2py*, n.d.), sendo esta uma *framework full-stack* desenvolvida para a construção de aplicações *web* baseadas em bases de dados.

Isto significa que as páginas *web* acedem diretamente à base de dados de forma a mostrar informação ao utilizador, dispensando a necessidade da relação cliente/servidor.

Com base neste pressuposto, é possível compreender que os padrões de migração analisados neste documento não se enquadram no contexto atual dada a impossibilidade de redirecionar os pedidos efetuados pelo cliente para o novo serviço desenvolvido.

Visando o *rollout* do novo serviço, a "estratégia de migração" adotada passou por garantir que a informação existente na base de dados do monólito está congruente com a informação existente no novo serviço de ASNs. Esta congruência de dados é conseguida através do consumo de todos os eventos referentes a mudanças de estado de uma ASN, aproveitando também uma das principais vantagens de uma arquitetura *event-driven.*

Desta forma, o monólito deve:

1. Consumir **ASNCreatedEvent** de forma a poder criar uma *PurchaseOrder* com a mesma informação.
2. Consumir **ASNClosedEventd** de forma a atualizar uma PurchaseOrder já criada, de acordo com os *items* recebidos.
3. Consumir **ASNCancelledEvent** de forma a cancelar uma PurchaseOrder já criada.

Para além do consumo destes três eventos, assegurou-se também a remoção da antiga página *web* que permitia a criação de "PurchaseOrders". Com esta remoção garante-se que o único ponto de entrada desta entidade seria o consumo de eventos provenientes do novo serviço. Desta forma, está assegurada não só a congruência de dados entre diferentes bases de dados, mas também a interoperabilidade entre as diferentes plataformas visto que os antigos processos que ainda utilizam a base de dados do monólito como fonte da verdade podem continuar a fazê-lo até que o processo de migração esteja concluído.

Todos estes eventos são produzidos através da conclusão do **RF#3**. O desenvolvimento da lógica associada ao consumo destes eventos está fora do contexto deste requisito.

6 Experimentação e avaliação

Nesta secção é descrita a hipótese de investigação, identificados os indicadores de avaliação e as fontes de informação, descrita a metodologia de avaliação para esses mesmos indicadores e, por fim, a aplicação dos métodos descritos de forma a avaliar a completude dos indicadores.

6.1 Hipótese de investigação

Como já referido neste documento, este projeto tem como principal objetivo dar início ao processo de atualização, restruturação e migração de um sistema maioritariamente monolítico para um sistema orientado a microsserviços, ajustando também os conceitos e paradigmas ao novo contexto, sendo exatamente esta a hipótese de investigação identificada.

O desenvolvimento dos novos microsserviços têm como pressuposto alterações quer na tecnologia adotada, que mudará de Python para .NET (C#), quer no padrão arquitetural a utilizar. Os novos serviços desenvolvidos devem assegurar o funcionamento da plataforma atual durante o processo de migração e irão também integrar a nova plataforma ECL.

6.2 Indicadores e fontes de informação

De acordo com a hipótese identificada, enumeram-se os seguintes indicadores:

- **Interoperabilidade**: Este indicador pretende avaliar se o objetivo referente à manutenção da plataforma Spoke foi atingido.
- **Velocidade de entrega**: Este indicador pretende avaliar se o objetivo relativo a melhorias no processo de desenvolvimento resultou numa maior velocidade de entrega de *story points*.

- **Manutenibilidade**: Este indicador pretende avaliar se objetivo relativo ao desenvolvimento de um projeto *template* contribuiu para a manutenibilidade dos serviços desenvolvidos.
- **Análise arquitetural**: Este indicador pretende avaliar se a arquitetura desenvolvida correspondeu às necessidades identificadas (segregação de responsabilidades de domínio e de equipa).

6.3 Metodologia de avaliação

Após a identificação dos indicadores, é necessário agora perceber como se pode avaliar a sua completude.

Os indicadores interoperabilidade, manutenibilidade e análise arquitetural serão avaliados através do desenvolvimento de um questionário que será respondido por um grupo de colaboradores que tomaram parte ativa neste projeto.

O indicador velocidade de entrega será avaliado através da análise de relatórios. Cada relatório será relativo a um único sprint e nele constará informação relativa à disponibilidade da equipa e respetivos *story points* entregues, tendo por base sempre a mesma história de referência.

6.4 Avaliação

Esta secção tem como objetivo aplicar os métodos de avaliação definidos na secção anterior de forma a poder avaliar os indicadores definidos.

6.4.1 *Questionário*

De forma a poder avaliar os indicadores referentes à interoperabilidade, manutenibilidade e análise arquitetural, foi desenvolvido um questionário que, como público-alvo, terá um grupo de colaboradores que tomaram parte ativa no projeto, sendo que, na sua grande maioria, são desenvolvedores de *software.* O questionário foi elaborado com a ajuda da ferramenta *Microsoft Forms* (*Microsoft Forms Help & Learning*, n.d.) e estará exposto nos anexos deste documento.

Este questionário está dividido em 4 secções distintas:

1. **Perfil do público-alvo**: parte introdutória do questionário, com um total de 8 perguntas cujo objetivo é dar contexto em relação ao público-alvo. São abordados tópicos como: cargo atual, anos de experiência, experiência em migrações arquiteturais, experiência em arquiteturas orientadas a microsserviços e dificuldades encontradas neste tipo de arquiteturas.

2. **Interoperabilidade**: segunda secção do questionário, com um total de 4 perguntas referentes ao indicador "interoperabilidade", cujo objetivo é compreender a completude do objetivo referente a garantir o funcionamento da antiga plataforma durante o processo de migração. São abordados tópicos como grau de completude, grau de dificuldade e estratégia escolhida.
3. **Manutenibilidade**: terceira secção do questionário, com um total de 4 perguntas referentes ao indicador "manutenibilidade", cujo objetivo é analisar de que forma o projeto *template* contribuiu para a manutenibilidade dos serviços criados. São abordados diferentes tópicos de forma a compreender de que forma o público-alvo perceciona o projeto *template*, vantagens do mesmo e frequência de utilização.
4. **Análise arquitetural**: última secção do documento, com um total de 2 perguntas referentes ao indicador "análise arquitetural", cujo objetivo é compreender se a arquitetura desenvolvida satisfaz as necessidades identificadas (segregação de responsabilidade de domínio e equipa).

Este questionário foi enviado a um total de dezanove colaboradores divididos em três diferentes equipas. Entre os dezanove colaboradores, obtiveram-se dezasseis respostas, em que o tempo médio de conclusão foi de três minutos e oito segundos como é possível visualizar na seguinte imagem:

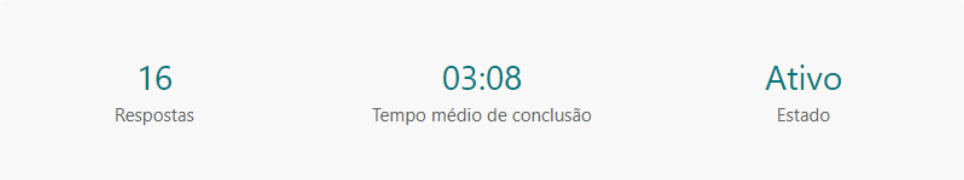


Figura 63 - Sumário do questionário

Relativamente à secção que pretende contextualizar o público-alvo, a primeira pergunta consiste em perceber o *background* do colaborador de forma a compreender se este faz parte dos membros que permaneceram na equipa após a compra da Maersk:

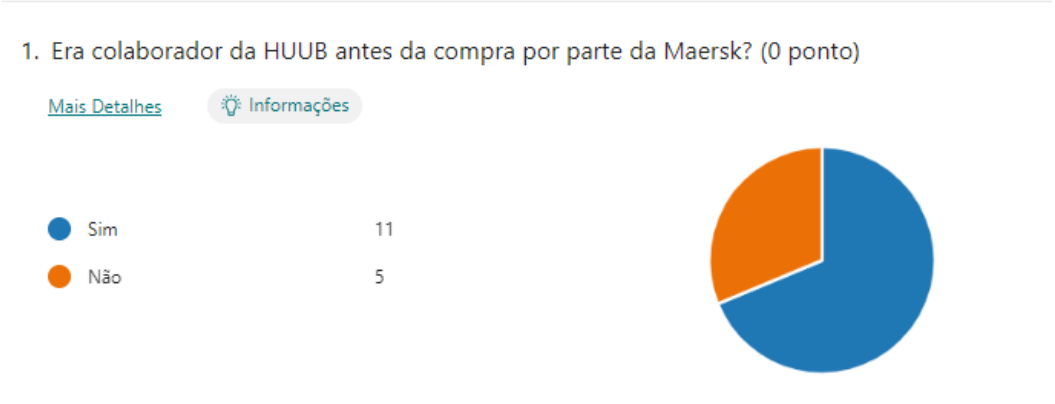


Figura 64 - Informação relativa ao percurso do público-alvo

Com base na figura 64 é possível compreender que 11 dos 16 colaboradores mantiveram-se no projeto (cerca de 69%), sendo que os restantes colaboradores foram contratados após o início do mesmo.

As próximas 2 questões são relativas ao cargo e experiência dos colaboradores:

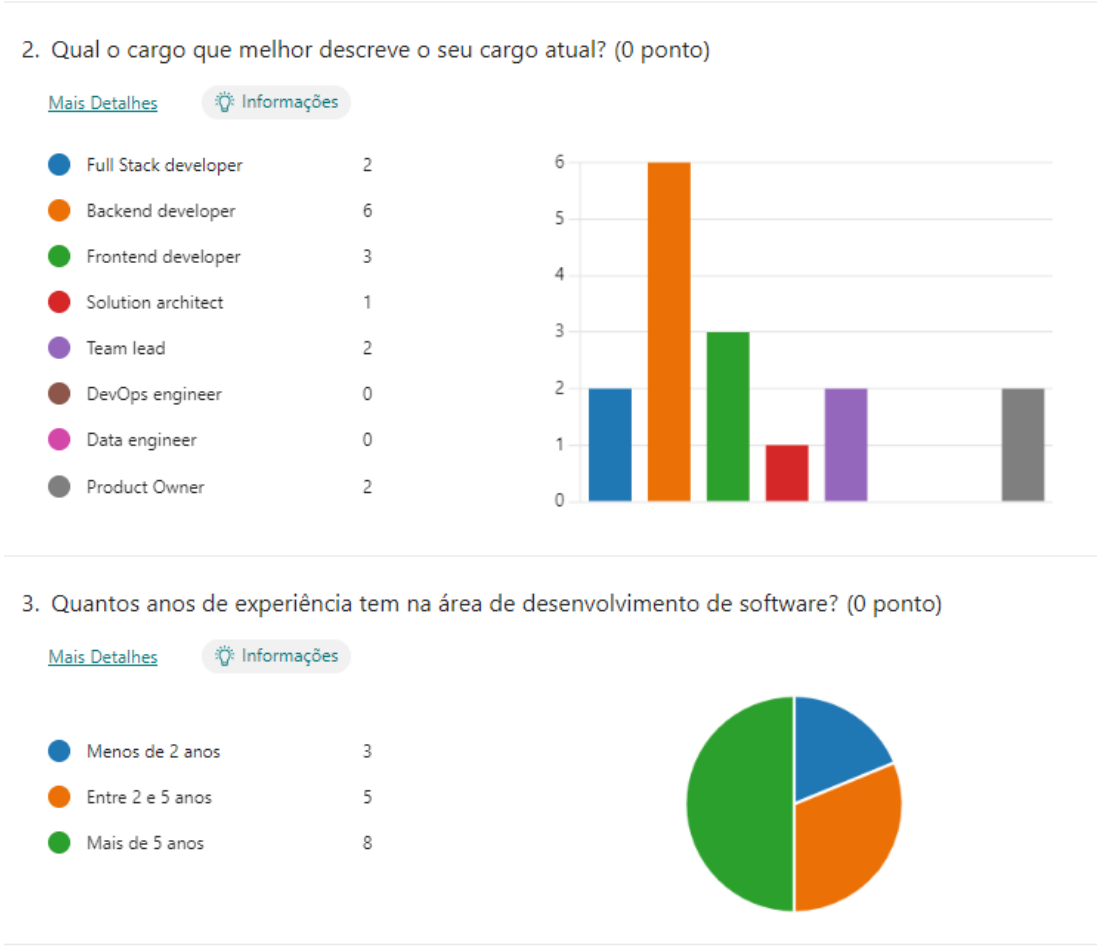


Figura 65 - Informação relativa à cargo e experiência do público-alvo

É possível analisar que a maior percentagem de colaboradores descreve o seu cargo atual como "*backend developer".* Para além disso, pode-se verificar também uma grande variedade de cargos nas respostas obtidas, sendo que não se obteve resposta apenas dos cargos "*DevOps Engineer"* e "*Data Engineer*". Esta variedade é importante para que se obtenham diferentes pontos de vista sobre a mesma questão.

Também é possível compreender apenas 3 dos 16 (cerca de 19%) colaboradores apresenta uma experiência profissional inferior a 2 anos e que exatamente 50% dos colabores apresentam uma experiência profissional superior a 5 anos. Estes dados são relevantes tendo em conta a complexidade associada ao projeto atual.

A próxima questão está diretamente relacionada com o novo "mundo" apresentado pela Maersk, visto que, como já referido neste documento, esta apresenta algumas restrições

tecnológicas. Tendo em conta que, como visualizado na estatística gerada pela primeira pergunta deste questionário, muitos colaboradores fizeram a transição HUUB para Maersk, faz sentido compreender de que forma estas restrições tecnológicas tiveram influência nestes mesmos colaboradores:

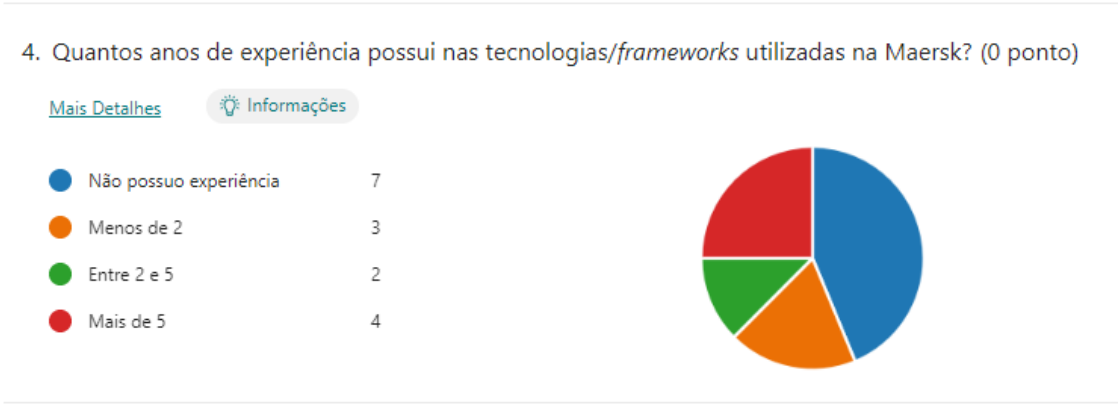


Figura 66 - Informação relativa à experiência tecnológica do público-alvo

Com base nesta imagem pode-se concluir que 7 dos 16 (cerca de 44%) inquiridos não possuem qualquer experiência naquelas que são as tecnologias impostas. Para além disso, 3 dos 16 colaboradores (cerca de 19%), apenas possui uma experiência marginal inferior a 2 anos. Estas 2 percentagens perfazem um total de 63% com pouca ou nenhuma experiência, valor este que, no início do projeto, afetou negativamente o tempo associado ao tempo de ciclo de entrega de *software*.

Dando continuidade a perguntas relativas à experiência tecnológica, a próxima pergunta tem por base compreender a experiencia associada à construção de sistemas cuja arquitetura está orientada a microsserviços:

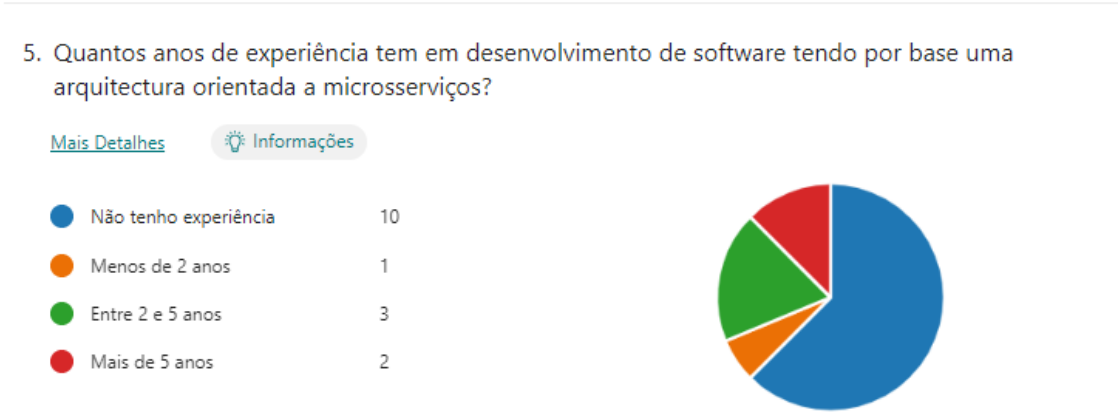


Figura 67 - Informação relativa à experiência arquitetural do público-alvo

Analisando a figura 67, é possível compreender que 10 dos 16 (cerca de 63%) colaboradores inquiridos não possuem qualquer experiência em sistemas que têm por base uma arquitetura

orientada a microsserviços. Este valor corresponde exatamente ao dobro do valor referente à percentagem de colaboradores que possui experiência superior a 2 anos.

Os inquiridos que não responderam “Não tenho experiência” na pergunta ilustrada na figura 67, foram inquiridos também em relação à complexidade associada ao desenvolvimento destes mesmos sistemas, obtendo-se a seguinte estatística:

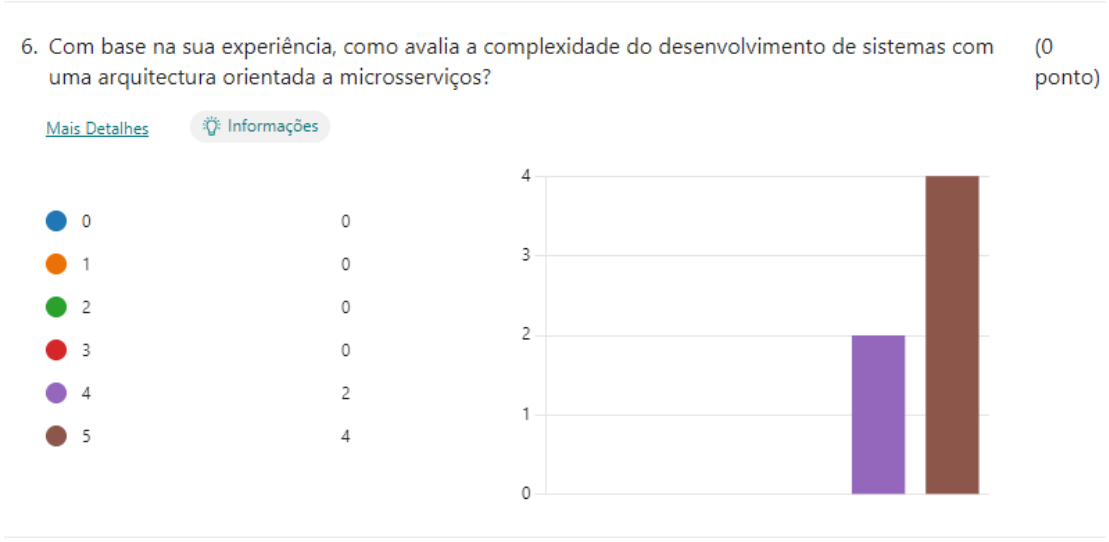


Figura 68 - Informação relativa à complexidade associada ao desenvolvimento de arquiteturas orientadas a microsserviços

Dos 6 inquiridos que tiveram acesso a esta questão, todos eles atribuíram uma elevada nota de grau de complexidade a estes sistemas.

A seguinte questão pretende compreender a experiência do público-alvo relativamente a processos de migração arquitetural de um sistema monolítico para um sistema orientado a microsserviços:

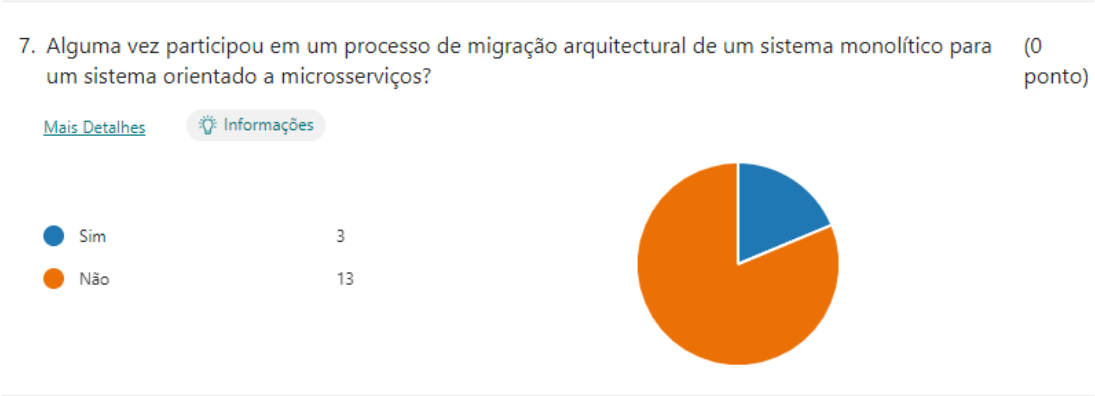


Figura 69 - Informação relativa à experiência em migrações do público-alvo

É possível verificar que a grande maioria dos inquiridos (cerca de 85%) nunca participou em nenhum processo de migração arquitetural.

A informação ilustrada pelas figuras 67, 68 e 69 demonstram que a equipa para além de ter pouca experiência em sistemas cuja arquitetura é orientadas a microsserviços e considerar estes sistemas complexos, também nunca esteve envolvida em um processo de tamanha complexidade. Este facto é ilustrativo do desafio encontrado durante o desenvolvimento deste projeto.

A última pergunta da primeira secção é apenas inquirida aos colaboradores que responderam "Sim" à questão ilustrada na figura 69 e pretende compreender as principais dificuldades encontradas durante estes processos:

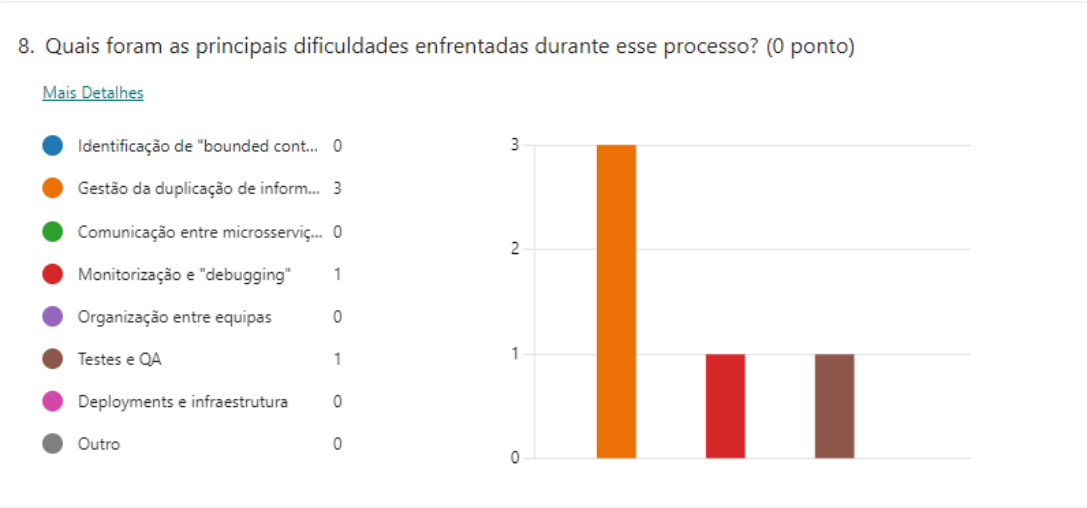


Figura 70 - Informação relativa às principais dificuldades associadas a migrações arquiteturais

Apesar de o número de respostas ser reduzido, podemos verificar que todos os colaboradores que responderam a esta questão identificaram "Gestão da duplicação de informação entre os vários serviços" como uma das principais dificuldades encontradas. Para além desta opção, as opções "Monitorização e *debugging*" e "Testes e QA" também foram identificadas como uma das principais dificuldades.

A figura 70 ilustra os dados recolhidos na questão número 8, questão esta que dá por terminada a secção relativa ao perfil do público-alvo. As próximas 4 questões são relativas ao indicador **interoperabilidade**, no qual se pretende compreender a completude do objetivo referente a garantir o funcionamento da antiga plataforma durante o processo de migração.

A primeira pergunta desta secção pretende analisar o grau de completude deste objetivo através de uma avaliação numérica de 0 a 5, onde zero representa a pior avaliação possível e o cinco representa que o objetivo foi totalmente atingido. De forma a auxiliar os inquiridos, todas as secções apresentam uma breve descrição do objeto em avaliação.

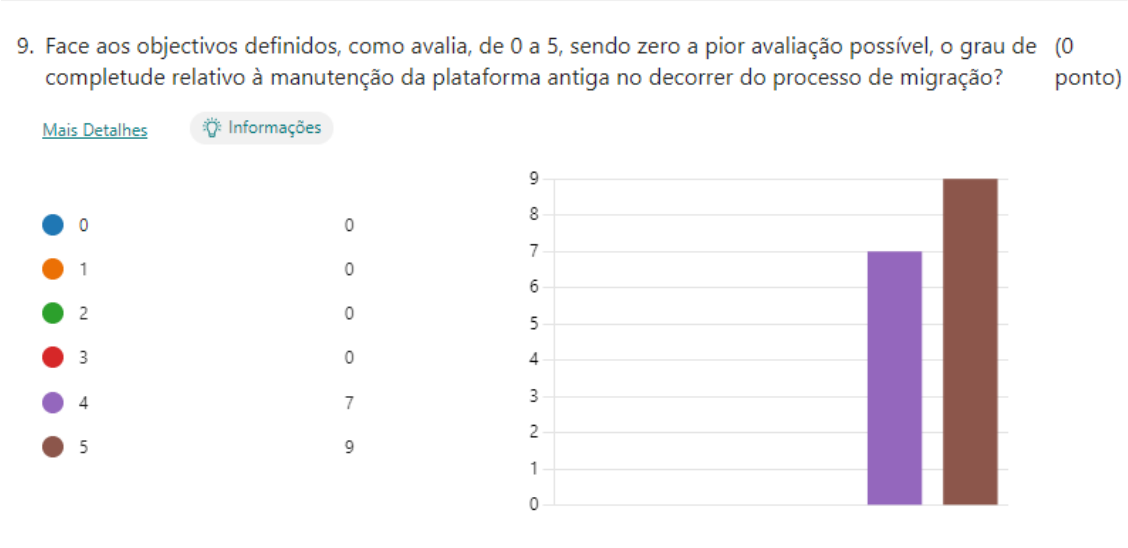


Figura 71 - Informação relativa ao grau de completude de interoperabilidade

Com base no gráfico acima ilustrado é possível perceber que todos os colaboradores atribuíram pelo menos a nota 4, sendo que 9 dos 16 (cerca de 57%) atribuíram nota máxima.

No seguimento desta questão pretende-se compreender o nível de dificuldade associado à manutenção da antiga plataforma, onde, novamente de 0 a 5, sendo que zero representa uma dificuldade baixa e cinco representa uma grande dificuldade:

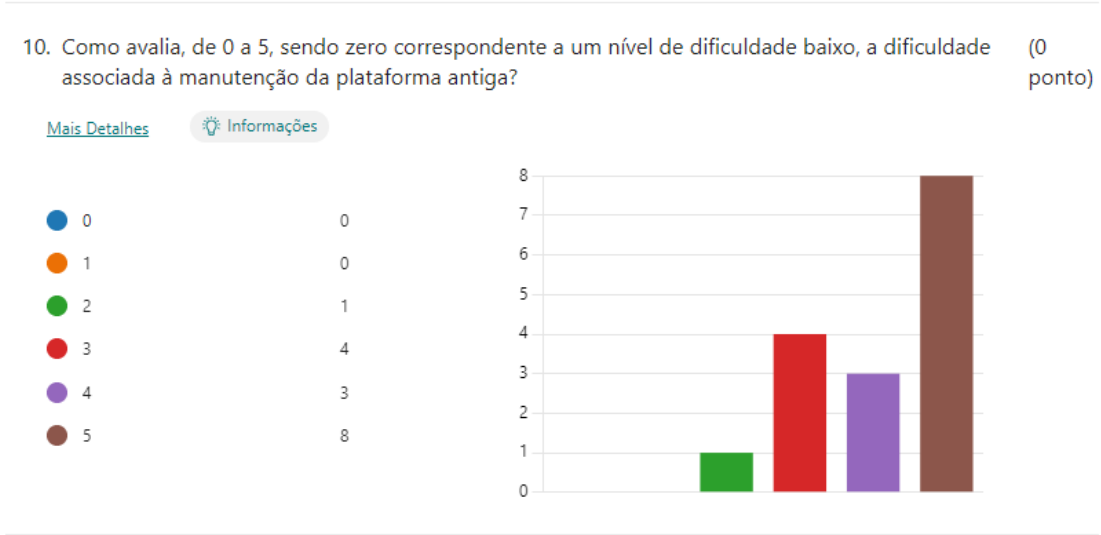


Figura 72 - Informação relativa ao grau de dificuldade de interoperabilidade

Apesar das respostas obtidas estarem distribuídas, é possível compreender que exatamente 50% dos inquiridos atribuiu nota máxima e que 75% inquiridos atribuíram uma nota superior ou igual a 4. Numa outra perspectiva, cerca de 31% dos inquiridos consideram a dificuldade deste objetivo relativamente baixa.

Continuando no tópico relativo à dificuldade deste objetivo, a seguinte figura ilustra a terceira questão desta secção, que tinha como finalidade compreender as principais dificuldades em garantir a interoperabilidade desejada:

Figura 73 - Informação relativa às dificuldades encontradas para garantir a interoperabilidade

Nesta questão, a grande maioria (75%) atribuiu como principal dificuldade a opção “Limitações tecnológicas relativas à plataforma antiga”. Para além desta opção praticamente consensual, as operações “diferenças de processos” e “diferenças nas tecnologias utilizadas” obtiveram, cada uma, dois votos.

Como última questão desta secção, pretendeu-se compreender de que forma os inquiridos avaliam a estratégia escolhida para garantir a interoperabilidade. Novamente, utilizando uma escala de 0 a 5, onde zero representa que a escolha não foi de todo adequada e o cinco representa que a escolha foi perfeitamente adequada:

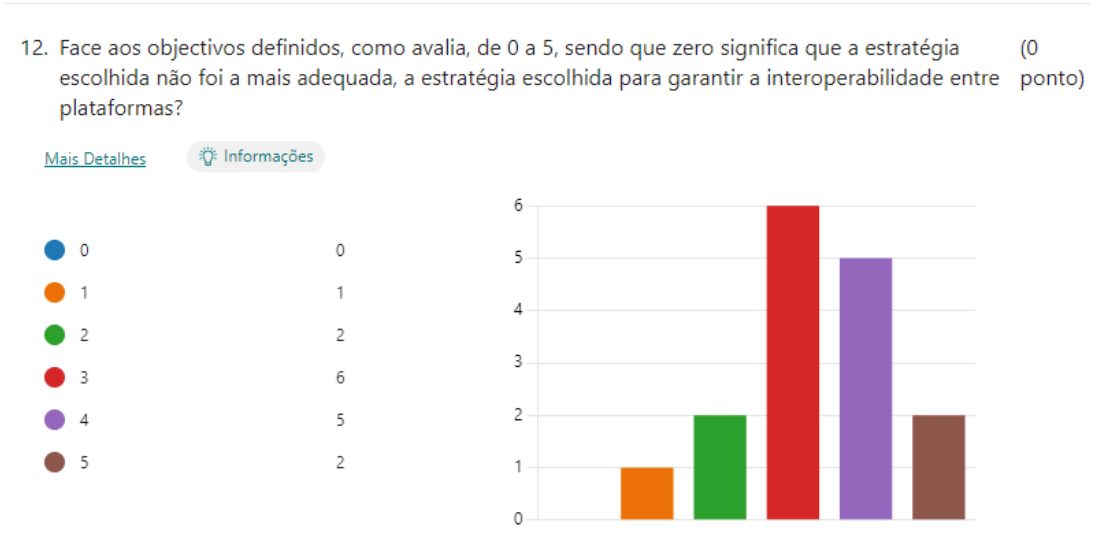


Figura 74 - Informação relativa à adequação da estratégia adotada

Como é possível constatar, esta questão foi das menos unânimes, sendo que, exceptuando a nota 0, todas tiveram votos. Mesmo assim, a resposta mais escolhida, nota 3, foi a que obteve mais votos, correspondendo a 38% dos votos totais. Numa outra perspectiva, cerca de 81% dos inquiridos atribuíram uma nota superior ou igual a 3, enquanto apenas 19% dos inquiridos atribuiu uma nota negativa.

Dando por encerrada secção relativa à interoperabilidade, as próximas 4 questões visam analisar de que forma o projeto *template* contribuiu, não só, mas também, para a manutenibilidade dos serviços criados.

A primeira pergunta utiliza, mais uma vez, uma escala de 0 a 5, onde zero representa a pior avaliação possível, de forma a avaliar o grau de completude relativo ao desenvolvimento do projeto *template:*

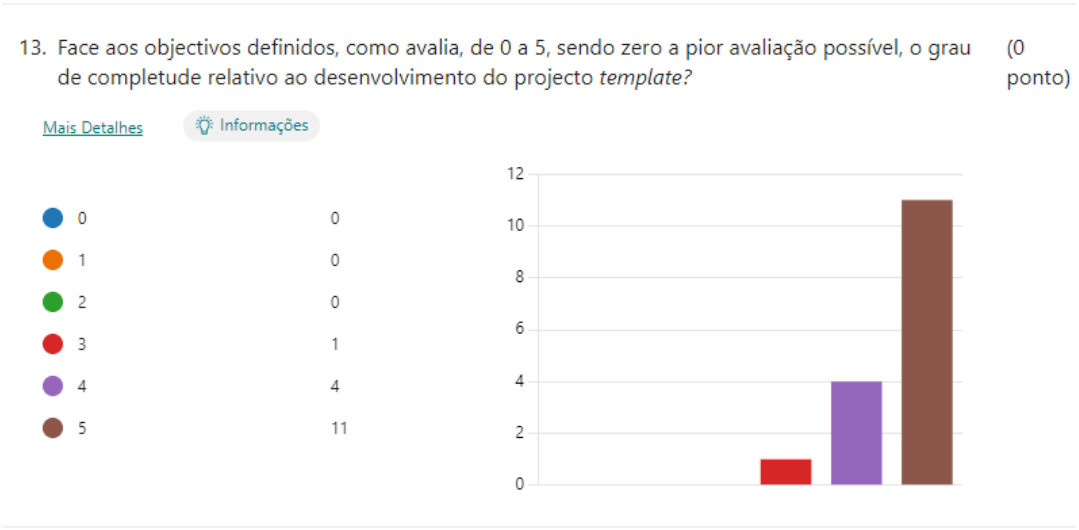


Figura 75 - Informação relativa ao grau de completude do projeto *template*

Cerca de 69% dos colaboradores atribuíram a nota máxima à completude do projeto *template*, sendo que, 15 dos 16 (cerca de 94%) atribuíram uma nota superior ou igual a 4.

A próxima questão é de escolha múltipla e dá seguimento àquela ilustrada na figura 75, de forma a compreender quais as principais vantagens encontradas pelos colaboradores que utilizaram o projeto *template*:

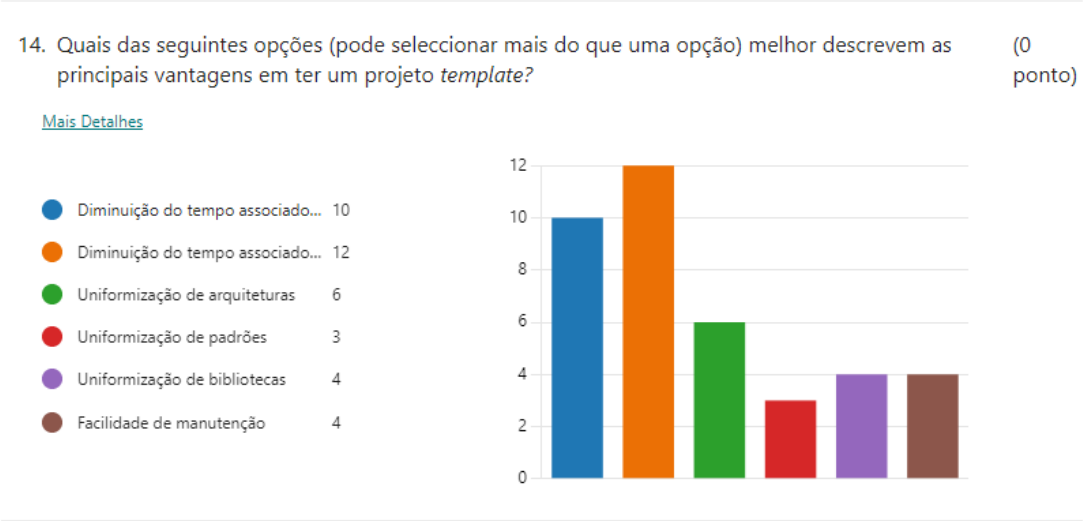


Figura 76 - Informação relativa às vantagens do projeto *template*

Com esta questão, utilizando um modelo de múltipla escolha sem limite de respostas (todas as opções podem ser selecionadas), pretende-se perceber quais foram as principais vantagens encontradas pelos colaboradores durante a utilização do projeto *template.* É possível compreender que cerca de 63% dos colaboradores responderam com "Diminuição do tempo associado à criação de um novo serviço" e que 75% responderam com "Diminuição do tempo associado à criação das pipelines de CI/CD para um novo serviço". Por outro lado, apenas 25% dos colaboradores selecionaram "Facilidade de manutenção" como uma das vantagens identificadas.

A seguinte figura ilustra a terceira pergunta desta secção, onde se pretende analisar a frequência com que este projeto é utilizado:

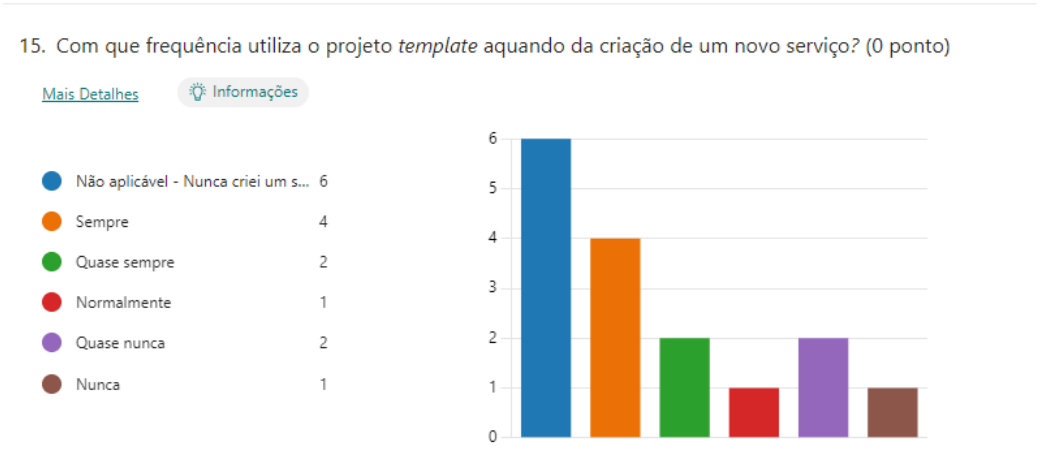


Figura 77 - Informação relativa à frequência de utilização do projeto *template*

De todos os inquiridos, pode-se verificar que cerca de 38% optaram por abster-se de atribuir uma frequência por não consideraram aplicável. Por outro lado, também cerca de 38% votaram

“Quase sempre” ou “Sempre”, sendo que apenas 19% dos inquiridos acredita que a frequência de utilização deste projeto é “Quase nunca” ou “Nunca”.

A última questão desta secção do questionário pretende compreender qual a avaliação atribuída pelos colaboradores à escolha arquitetural, padrões e bibliotecas escolhidas para o desenvolvimento dos novos microsserviços:

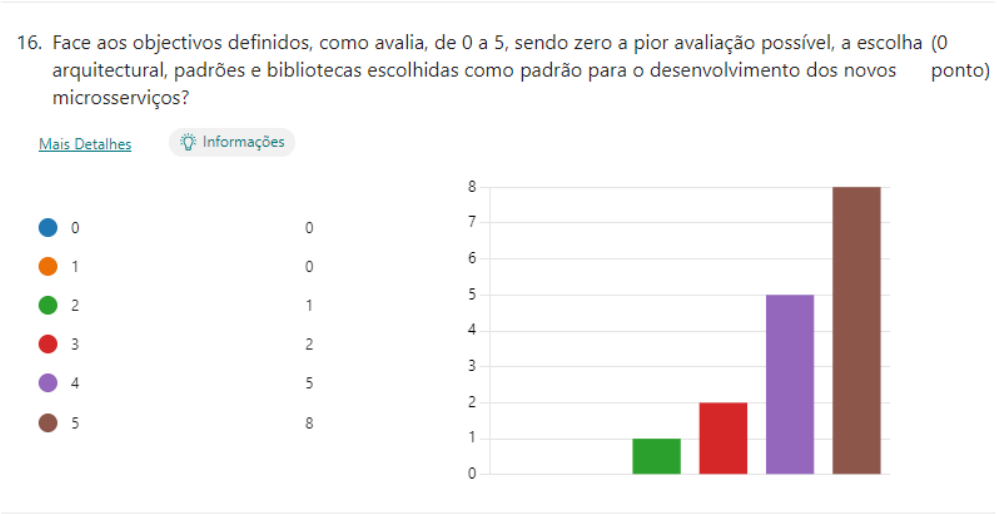


Figura 78 - Informação relativa à avaliação das escolhas efetuadas

Para finalizar esta secção, é possível constatar que exatamente 50% dos colaboradores inquiridos atribuíram nota máxima às escolhas já referidas. É de salientar que 13 dos 16 (cerca de 81%) colaboradores inquiridos atribuiu uma nota igual ou superior a 4. Por outro lado, apenas 1 dos 16 colaboradores atribuiu a nota de 2.

As próximas 2 questões correspondem à secção relativa à **análise arquitetural**, onde, através de uma escala de 0 a 5, em que zero representa a pior avaliação possível, se pretendia perceber se a arquitetura desenvolvida satisfaz as necessidades identificadas (segregação de responsabilidade de domínio e equipa):

Figura 79 - Informação relativa à avaliação da satisfação das necessidades identificadas

A primeira pergunta visa compreender de que forma a arquitetura desenvolvida contribuiu para a segregação de responsabilidades de domínio. É possível constatar que 38% dos inquiridos atribuíram a nota máxima. É de salientar também que cerca de 87% dos inquiridos atribuíram uma nota superior ou igual a 4, enquanto apena 13% dos mesmos atribuíram notas relativamente baixas.

Em relação à segunda e última questão desta secção, pretendia-se constatar de que foram os colaboradores avaliam a segregação de responsabilidades entre equipas resultante da arquitetura desenvolvida. Apenas cerca de 13% dos colaboradores atribuíram a nota 3, podendo-se desta forma perceber que 87% dos colaboradores estão satisfeitos com este quesito.

6.4.2 *Relatórios por sprint*

De forma a avaliar o indicador **velocidade de entrega,** recolheu-se dados relativos à entrega de *story points* no decorrer de todo o projeto. O objetivo seria, através de uma melhoria e automatização significativa dos processos com base nos conceitos teóricos recolhidos e aplicados gradualmente, verificar um aumento também gradual de *story points* entregues.

É importante salientar que os dados recolhidos têm por base a mesma história de referência, de forma a evitar discrepâncias entre os valores recolhidos. O número de colaboradores foi também constante durante todo o processo. A única variável identificada é a disponibilidade da equipa, que varia consoante o número de ausências por *sprint.*

A seguinte tabela representa, por sprint, a respetiva entrega em número de *story points* e disponibilidade da equipa, em percentagem. A última coluna representa a entrega de *story points* caso a disponibilidade da equipa fosse 100% (estimativa calculada com base em *story points* entregues e disponibilidade da equipa, com base na utilização da "regra de três simples" (*Regra de Três Simples*, n.d.)).

Tabela 12 - Tabela de *story points* entegues por *sprint*

Data início/final do sprint	*Story points*	Disponibilidade da equipa (%)	*Story points* a 100%
30/01/2023 - 10/02/2023	38	94	40
13/02/2023 - 24/02/2023	32	92	35
27/02/2023 - 10/03/2023	52	100	52
13/03/2023 - 24/03/2023	44	100	44
27/03/2023 - 07/04/2023	37	86	43
10/04/2023 - 21/04/2023	62	100	62
24/04/2023 - 05/05/2023	36	94	38
08/05/2023 - 19/05/2023	42	98	43
22/05/2023 - 02/06/2023	38	88	43
05/06/2023 - 16/06/2023	40	86	47

De seguida, com auxílio da ferramenta *Microsoft Excel*, utilizou-se os *story points* calculados de forma a construir um gráfico do qual se pudesse verificar o esperado aumento gradual.

Para tal, depois da construção do respetivo gráfico e, de forma a aplicar o método de regressão linear (*Regression Analysis | Real Statistics Using Excel*, n.d.) com o objetivo de analisar a relação entre o número de *story points* entregues e o decorrer do projeto, utilizou-se uma das funcionalidades disponibilizadas pelo *Microsoft Excel*, a função *"trendline"*.

Esta função, para além de traçar a reta que melhor representa todos os pontos do gráfico através da utilização do método dos quadrados mínimos (*Method of Least Squares | Real Statistics Using Excel*, n.d.), também calcula o declive e o R^2 da mesma (*Add a Trendline in Excel (In Easy Steps)*, n.d.).

De forma a compreender se a linha de tendência gerada está ajustada aos dados, o valor de R^2 deve ser próximo de 1. Outro valor importante obtido através da regressão linear é o declive da reta que, ao ser positivo, revela uma tendência positiva entre as variáveis, isto é, que a tendência é o valor de *story points* entregues aumentar.

Desta forma, obteve-se o seguinte gráfico e respetiva *trendline:*

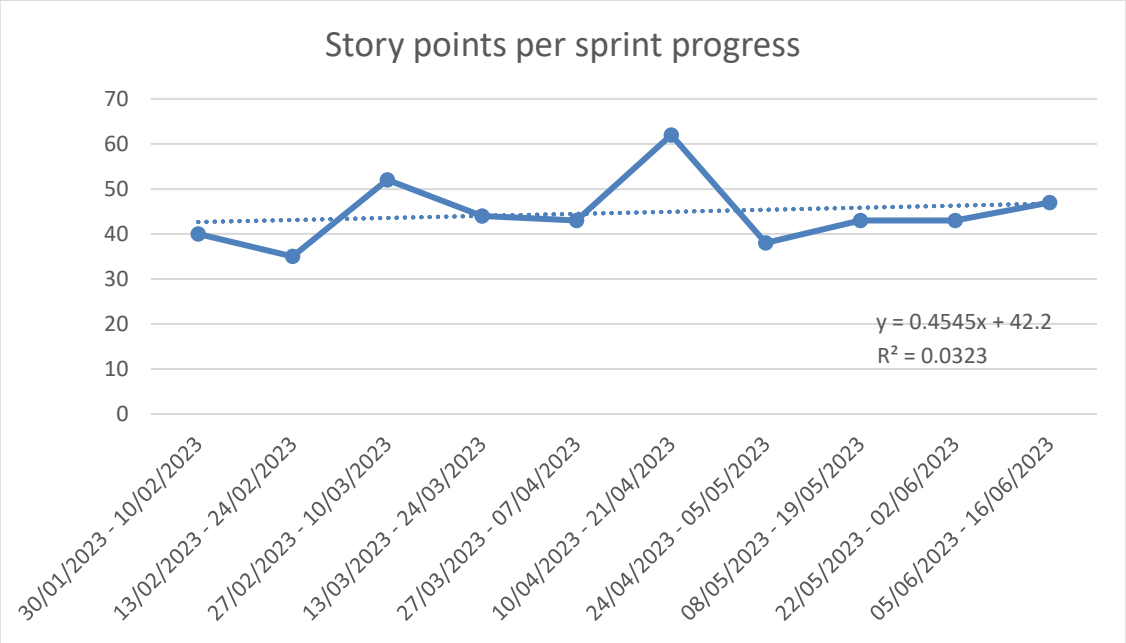


Figura 80 - Gráfico representativo do número de *story points* por *sprint*

Como é possível observar, o valor de R^2 gerado não é perto de 1, o que significa que a reta gerada não representa com precisão a grande parte dos valores presente no gráfico. O facto do valor de R^2 ser tão baixo pode dever-se a variadas razões, entre as quais:

- **Variação aleatória:** a informação recolhida em relação aos *story points* entregues apresenta uma certa aleatoriedade, o que pode originar uma reta que não consiga representar com precisão os valores apresentados
- **Relação não linear:** é um facto que para este caso específico foi utilizada uma função linear. Porém, é possível que esta não seja a que melhor represente a relação entre as variáveis representadas, havendo múltiplas alternativas, como funções logarítmicas, exponenciais etc.
- ***Outliers*:** a presença de excedentes, considerados valores significativamente distantes dos restantes, pode afetar o ajuste da *trendline.*

Analisando o gráfico representado, é possível verificar uma certa aleatoriedade nos valores, não sendo possível detectar padrões claros que constatem uma relação direta entre as variáveis

representadas. Dentro desta "aleatoriedade", é possível também constatar a presença de um *outlier*, sendo este o valor de 62 *story points* entregues durante o *sprint* que ocorreu entre os dias 14/04/2023 e 21/04/2023.

Apesar de um valor não muito positivo de R^2, o valor registado para o declive da reta é 0.454, valor este que revela uma inclinação positiva. Um declive positivo, apesar do seu valor baixo, pode ser indicador de uma relação positiva entre as variáveis identificadas, indicando, desta forma, que o valor de *story points* entregues por *sprint* têm tendencia a aumentar para futuras *sprints.*

6.5 Considerações finais

Com base nos indicadores recolhidos e na avaliação efetuada através da elaboração de um questionário, seguido da respetiva análise, e análise dos relatórios por sprint, consegue-se retirar conclusões acerca de múltiplos tópicos deste projeto.

A primeira secção do questionário permitiu perceber que os colaboradores que contribuíram ativamente para este projeto apresentavam pouca experiência nas tecnologias utilizadas na Maersk. Para além disso, apresentavam também uma visível baixa experiência em desenvolvimento de sistemas orientados a microsserviços, considerando-os também sistemas de elevada complexidade. Em adição, é possível constatar que apenas uma reduzida amostra dos colaboradores já participou em processos que envolvessem uma migração de uma arquitetura monolítica para uma orientada a microsserviços.

Relativamente à segunda secção do questionário, referente ao indicador **interoperabilidade**, é possível analisar que todos os colaboradores atribuíram um elevado grau de sucesso à manutenção da plataforma antiga durante o processo de migração. Este facto é de elevada relevância tendo em conta que a grande maioria dos colaboradores considerou que a manutenção da plataforma foi de elevada dificuldade. Este elevado grau de dificuldade atribuído teve como principal motivo, segundo os dados obtidos, as limitações tecnológicas relativas à plataforma antiga. Por outro lado, a estratégia escolhida para garantir esta interoperabilidade não foi considerada a melhor pela maioria dos colaboradores inquiridos. Com base nestas respostas, a avaliação do sucesso da plataforma é positiva.

Em relação ao indicador **manutenibilidade**, abordado na terceira secção do questionário, que pretendia analisar o contributo do projeto *template*, é possível verificar que a maioria dos inquiridos atribuiu um elevado grau de sucesso ao desenvolvimento deste projeto. Para além disso e, de acordo com os dados recolhidos, a grande maioria dos inquiridos refere que as principais vantagens do projeto *template* são a diminuição de tempo, quer referente à criação de novos microsserviços, quer à criação de pipelines de integração. Por outro lado, apenas uma reduzida porção dos inquiridos refere "manutenibilidade" como uma vantagem do desenvolvimento deste projeto. Este facto pode ser justificado com o pouco tempo de utilização que este tem, não sendo possível tirar ainda dados concretos em relação a este fator em específico. Ainda assim, com base em toda a informação recolhida e, tendo em conta a

imaturidade deste projeto, é possível considerar o *contributo* do projeto *template* como positivo.

O último indicador avaliado via questionário, **análise arquitetural,** pretendia avaliar se a arquitetura desenvolvida correspondia às necessidades de segregação de responsabilidades de domínio equipa. Ambas as necessidades identificadas tiveram atribuídos elevados graus de sucesso pela grande maioria dos inquiridos. Desta forma, é possível afirmar que a análise arquitetural satisfez as necessidades identificadas com sucesso.

Por fim, o indicador **velocidade de entrega,** onde se pretende avaliar se o objetivo relativo a melhorias no processo de desenvolvimento resultou numa maior velocidade de entrega de *story points*. De acordo com os dados recolhidos por *sprint* e respetiva análise efetuada, é possível compreender que a tendência é positiva, apontando para um acréscimo no número de *story points* entregues por *sprint*. Desta forma, pode-se concluir que as melhorias no processo de desenvolvimento contribuíram com sucesso para uma maior velocidade de entrega.

7 Conclusões

Nesta secção são abordadas as conclusões através de uma análise aos objetivos concluídos face aos propostos e de uma reflexão acerca de trabalho futuro.

7.1 Análise aos objetivos

Este projeto consistia em dar início ao processo de atualização e restruturação de uma plataforma através de uma migração para uma arquitetura orientada a microsserviços.

De forma a alcançar essa mesma meta foram identificados e analisados um conjunto de objetivos. Resta agora efetuar uma análise a cada um deles, de forma a compreender o seu nível de completude.

7.1.1 *Segmentação do monólito*

Este objetivo tinha como finalidade dar início à segmentação do monólito existente por intermédio da realização uma análise ao sistema atual e início do processo de segmentação do módulo de OMS através da realização de um conjunto de requisitos.

Tendo em conta que tanto a análise ao sistema atual como todos os requisitos foram desenvolvidos com sucesso e que, devido a estes mesmos factos, foi possível dar-se início ao *rollout* do novo serviço, o objetivo foi cumprido com sucesso.

7.1.2 *Interoperabilidade*

Um dos principais objetivos passava por manter a aplicação Spoke funcional até que esteja completamente migrada. Este objetivo teve um elevado grau de dificuldade devido às tecnologias utilizadas para a construção da antiga plataforma. A *framework web2py* é uma *framework full-stack* desenvolvida para a construção de aplicações *web* baseadas em bases de dados. Isto significa, em poucas palavras, que a típica relação cliente/servidor não existe, havendo uma ligação direta entre a informação exibida nas páginas e a informação existente na base de dados.

Com base neste pressuposto, tornou-se inviável a utilização um dos passos fundamentais na maioria dos padrões de migração: o redireccionamento de tráfego.

Tendo em conta esta limitação, tirou-se partido de uma das principais vantagens de uma arquitetura *event-driven* e, através do consumo de todos os eventos produzidos pelo novo serviço construído, foi-se capaz de criar uma cópia fidedigna da informação existente na base de dados do novo serviço.

Desta forma, a antiga página encontra-se atualizada, em tempo real, até que uma nova página seja desenvolvida e esta se possa abandonar definitivamente.

Para além disso, os dados recolhidos durante a secção relativa à experimentação e avaliação da solução apontam para um elevado grau de sucesso, apesar do reconhecido elevado grau de dificuldade.

Sendo assim, pode-se afirmar que o objetivo relativo à interoperabilidade entre plataformas foi alcançado com sucesso.

7.1.3 *Estudo de diferentes abordagens*

Este objetivo tinha como finalidade garantir que eram analisadas e estudadas diferentes arquiteturas baseadas em microsserviços, sendo que a escolha entre as diferentes opções deveria ser baseada no resultado da aplicação de um método multicritério.

Para além disso, com este objetivo pretendia-se também o estudo e análise de diferentes abordagens referentes ao processo de migração, sendo que a abordagem escolhida deveria garantir o funcionamento da antiga plataforma.

Relativamente à parte deste objetivo referente a análise de diferentes arquiteturas: foi concluída com sucesso no sentido em que foram analisadas e desenvolvidas 2 arquiteturas, sendo que a arquitetura que prosseguiu para a fase de desenvolvimento foi aquela que melhor resultado apresentou após a aplicação do método multicritério AHP.

No que diz respeito à parte deste objetivo que refere o estudo e análise de diferentes abordagens ao processo de migração: foi concluída com sucesso visto que foram analisadas múltiplas abordagens e, após reflexão, concluiu-se que, devido às particularidades tecnológicas

já referenciadas neste documento, nenhuma correspondia às necessidades atuais. A abordagem escolhida, apesar de, com base nos dados recolhidos na secção de experimentação e avaliação, não ser considerada a mais adequada, foi a abordagem possível e contribuiu de forma preponderante para o alcançar de múltiplos objetivos.

7.1.4 *Resolução da dívida técnica*

Este objetivo foi definido para tentar colmatar alguma da dívida técnica associada à prática de desenvolvimento de software, visto terem sido identificadas inúmeras lacunas na plataforma existente: ausência de padrões, boas práticas de desenvolvimento de software e discrepância entre abordagens efetuadas pelas diferentes equipas.

De forma a tentar não voltar a cometer estes mesmos erros já identificados, analisaram-se e aplicaram-se boas práticas de desenvolvimento *software* através da definição de uma arquitetura base, padrões e bibliotecas a utilizar.

Com base nesta informação e, consciente que a resolução da divida técnica é um trabalho que requer constante atenção, este objetivo foi alcançado com sucesso.

7.1.5 *Melhorias no processo de desenvolvimento*

Com este objetivo pretendia-se alcançar um aumento de velocidade de entrega de funcionalidades, sendo que para alcançá-lo deveriam ser estudados e aplicados conceitos relativos ao processo de desenvolvimento de *software*.

De forma a completar este objetivo, foram analisados inúmeros tópicos associados a abordagens e qualidade, como por exemplo **QA** (*Quality Assurance*) e **QC** (*Quality Control*). Foram também estudados diversos tópicos associados ao tempo de ciclo da entrega de *software*, como *pipelines* de CI/CD, análise estática de código, *templates* e ainda monitorização automática de erros. Para finalizar este estudo, foram ainda abordados temas associados a modelos de qualidade e padrões de desenvolvimento, como *test-driven development* e *onion architecture.*

Todos estes conceitos foram depois aplicados, de forma gradual, durante a fase desenvolvimento, contribuindo de forma preponderante para, segundo a análise efetuada na secção “relatórios por sprint”, o aumento do número de *story points* entregues.

Com base no trabalho efetuado e nos dados recolhidos e analisados, este objetivo foi alcançado com sucesso.

7.1.6 *Uniformização de serviços*

Com este objetivo pretendia-se o desenvolvimento de um projeto *template* que contribuísse para a uniformização dos projetos, facilitando o processo de criação e manutenção dos mesmos. Para este desenvolvimento, deveria ter sido em conta padrões a utilizar, bibliotecas externas, etc.

Os dados recolhidos e analisados na secção "experimentação e avaliação" permitiram compreender que o projeto *template* contribuiu para inúmeros fatores, entre os quais: uniformização de arquiteturas, uniformização de padrões e uniformização de bibliotecas.

Tendo em conta estes mesmos dados e todo o tempo investido na análise de boas práticas de desenvolvimento de *software*, padrões a utilizar, análise e conceção de uma arquitetura a aplicar e posterior desenvolvimento do projeto *template*, pode-se afirmar que este objetivo foi alcançado com sucesso.

7.2 Trabalho futuro

Embora tenham sido alcançados com sucesso todos os objetivos propostos, é inconcebível pensar neste projeto como finalizado tendo em conta que este é apenas o início do processo de migração da plataforma. Para além disso, existe também a necessidade de melhora contínua dos processos atuais e, provavelmente, a inclusão futura de novas funcionalidades

Em relação ao projeto *template*, é possível interpretá-lo como a "versão 1.0", visto que este, apesar do *feedback* positivo recolhido durante o questionário efetuado da secção "experimentação e avaliação", ainda não foi utilizado tempo suficiente para se poder obter conclusões claras em relação à arquitetura escolhida, padrões adotados e bibliotecas selecionadas. Desta forma, prevê-se que este projeto sofra alterações em um futuro próximo.

Relativamente à criação do microsserviço resultante da segregação do módulo de OMS: o desenvolvimento do serviço "ecl-asn-service" foi um sucesso pois foi-se capaz de desenvolver todos os requisitos funcionais identificados, não abdicando de boas práticas de gestão, análise e desenvolvimento de *software*. Contudo, é possível compreender que alguns dos processos existentes ainda utilizam a base de dados do monólito como fonte de verdade. Desta forma, deve-se continuar os desenvolvimentos de forma a garantir que o novo serviço desenvolvido passe a ser a *source of truth* para e entidade ASN.

Por fim, o objetivo relativo à resolução da dívida técnica tende a ser um objetivo que exige trabalho constante, de forma a garantir que os serviços continuam com elevado nível de manutenibilidade, permitindo a adição de funcionalidades.

Referências

5 Biggest order fulfillment challenges that ecommerce companies face - Amazon Multi-Channel Fulfillment (MCF). (2022). https://supplychain.amazon.com/blog/5-biggest-order-fulfillment-challenges-ecommerce-companies-face

About A.P. Moller - Maersk | Maersk. (n.d.). Retrieved 11 February 2023, from https://www.maersk.com/about

Add a Trendline in Excel (In Easy Steps). (n.d.). Retrieved 15 June 2023, from https://www.excel-easy.com/examples/trendline.html

ASP.NET Core Integration | Quartz.NET. (2023). https://www.quartz-scheduler.net/documentation/quartz-3.x/packages/aspnet-core-integration.html

Domain events: Design and implementation | Microsoft Learn. (n.d.). Retrieved 28 May 2023, from https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/domain-events-design-implementation

dotnet restore command - .NET CLI | Microsoft Learn. (2023). https://learn.microsoft.com/en-us/dotnet/core/tools/dotnet-restore

Ecommerce Defined: Types, History, and Examples. (2022). https://www.investopedia.com/terms/e/ecommerce.asp

E-Commerce Logistics & Order Fulfilment Services | Maersk. (n.d.). Retrieved 11 February 2023, from https://www.maersk.com/supply-chain-logistics/e-commerce-logistics

Encrypted secrets - GitHub Docs. (n.d.). Retrieved 17 June 2023, from https://docs.github.com/en/actions/security-guides/encrypted-secrets

Fowler, M. (2014a). *BoundedContext*. https://martinfowler.com/bliki/BoundedContext.html

Fowler, M. (2014b). *BranchByAbstraction*. https://martinfowler.com/bliki/BranchByAbstraction.html

Fowler, M. (2014c). *DDD_Aggregate*. https://martinfowler.com/bliki/DDD_Aggregate.html

Fowler, M. (2014d). *Microservices*. https://martinfowler.com/articles/microservices.html

Fowler, M. (2014e). *StranglerFigApplication*. https://martinfowler.com/bliki/StranglerFigApplication.html

Fowler, M. (2019). *TechnicalDebt*. https://martinfowler.com/bliki/TechnicalDebt.html

Grafana: The open observability platform | Grafana Labs. (n.d.). Retrieved 17 June 2023, from https://grafana.com/

Hamilton, T. (2023). *Quality Assurance vs Quality Control – Difference Between Them*. https://www.guru99.com/quality-assurance-vs-quality-control.html

Hangfire – Background jobs and workers for .NET and .NET Core. (n.d.). Retrieved 6 June 2023, from https://www.hangfire.io/

Home | Product Blog • Sentry. (n.d.). Retrieved 18 February 2023, from https://blog.sentry.io/?gclid=Cj0KCQiA6LyfBhC3ARIsAG4gkF-pHR9c5VhOxrNvCHXUICl9p3ckPDrt4SwPO1MR_VXQgxYhMDHzNU8aAlmJEALw_wcB&utm_content=g&utm_term=sentry&utm_medium=cpc&utm_source=google&utm_campaign=19648478148

HTTP | MDN. (2022). https://developer.mozilla.org/en-US/docs/Web/HTTP

Install .NET on Windows - .NET | Microsoft Learn. (2023). https://learn.microsoft.com/en-us/dotnet/core/install/windows?tabs=net70

Introduction to Apache Kafka Partitions. (n.d.). Retrieved 28 May 2023, from https://developer.confluent.io/learn-kafka/apache-kafka/partitions/

JSON Web Tokens - jwt.io. (n.d.). Retrieved 28 May 2023, from https://jwt.io/

Kapoor, A. (2021). *Patterns to know before migrating your monolith to microservices | by Abhishek Kapoor | Level Up Coding*. https://levelup.gitconnected.com/patterns-to-know-before-migrating-your-monolith-to-microservices-72fcbcc7846e

Koen. (2001). *The PDMA ToolBook 1 for New Product Development - Google Livros*. https://books.google.pt/books?hl=pt-PT&lr=&id=kqX5EvT2U8AC&oi=fnd&pg=PA5&dq=Koen,+P.A.+et+al.+(2001)+%E2%80%9CFuzzy+FrontEnd:+Effective+Methods,+Tools,+and+Techniques&ots=8Lpo8bqMgd&sig=dEY-lbYAuVBRCyfgYu_F2k153tg&redir_esc=y#v=onepage&q&f=false

Mediator. (n.d.). Retrieved 25 May 2023, from https://refactoring.guru/design-patterns/mediator

Mediator Pattern com MediatR no ASP.NET Core | Blog TreinaWeb. (2020). https://www.treinaweb.com.br/blog/mediator-pattern-com-mediatr-no-asp-net-core

Method of Least Squares | Real Statistics Using Excel. (n.d.). Retrieved 15 June 2023, from https://real-statistics.com/regression/least-squares-method/

Microservices Adoption in 2020 – O'Reilly. (2020). https://www.oreilly.com/radar/microservices-adoption-in-2020/

Microservices vs Monolithic Architecture - What Should You Choose? | XME Advice. (2022). https://www.xme.digital/post/xme-advice-is-it-worth-moving-from-monolithic-to-microservices-architecture

Microsoft Forms help & learning. (n.d.). Retrieved 13 June 2023, from https://support.microsoft.com/en-us/forms

Modelo de qualidade externa e interna ISO/IEC 9126. | Download Scientific Diagram. (2019). https://www.researchgate.net/figure/Figura-5-Modelo-de-qualidade-externa-e-interna-ISO-IEC-9126_fig5_332822299

.NET CLI | Microsoft Learn. (2022). https://learn.microsoft.com/en-us/dotnet/core/tools/

Newman, S. (2019). *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith - Sam Newman - Google Livros*. https://books.google.com.br/books?hl=pt-PT&lr=&id=nNa_DwAAQBAJ&oi=fnd&pg=PP1&dq=monolith+to+microservices+sam+newman#v=onepage&q=monolith%20to%20microservices%20sam%20newman&f=false

Nicola, S. (n.d.-a). *ANÁLISE DE VALOR*.

Nicola, S. (n.d.-b). *ANÁLISE DE VALOR INESC-TEC*.

Npgsql Entity Framework Core Provider | Npgsql Documentation. (n.d.). Retrieved 15 May 2023, from https://www.npgsql.org/efcore/

O que é conteinerização? – Explicação sobre conteinerização – AWS. (n.d.). Retrieved 29 June 2023, from https://aws.amazon.com/pt/what-is/containerization/

O que é uma marca | Justiça.gov.pt. (2020). https://justica.gov.pt/Registos/Propriedade-Industrial/Marca/O-que-e-uma-marca

Object graph comparison - Fluent Assertions. (n.d.). Retrieved 14 May 2023, from https://fluentassertions.com/objectgraphs/

Paulovich, I. (2019). *FluentMediator*. https://github.com/ivanpaulovich/FluentMediator

Realistic Data Generation in .NET With Bogus - Code Maze. (n.d.). Retrieved 17 June 2023, from https://code-maze.com/data-generation-bogus-dotnet/

Regra de Três Simples. (n.d.). Retrieved 29 June 2023, from https://www.matematica.pt/util/calculadora-regra-3-simples.php

Regression Analysis | Real Statistics Using Excel. (n.d.). Retrieved 15 June 2023, from https://real-statistics.com/regression/regression-analysis/

Required Query String Parameters in ASP.NET Core - Code Maze. (n.d.). Retrieved 17 June 2023, from https://code-maze.com/aspnetcore-required-query-string-parameters/

Revolutionizing the Fashion Industry - HUUB. (n.d.). Retrieved 11 February 2023, from https://www.thehuub.co/about/

Rich, N., & Holweg, M. (2000). *Report produced for the EC funded project INNOREGIO: dissemination of innovation and knowledge management techniques V V VA A AL L LU U UE E E A A AN N NA A AL L LY Y YS S SI I IS S S V V VA A AL L LU U UE E E E E EN N NG G GI I IN N NE E EE E ER R RI I IN N NG G G*.

Richardson, C. (2018a). *Circuit Breaker*. https://microservices.io/patterns/reliability/circuit-breaker.html

Richardson, C. (2018b). *Command Query Responsibility Segregation (CQRS)*. https://microservices.io/patterns/data/cqrs.html

Richardson, C. (2018c). *Database per service*. https://microservices.io/patterns/data/database-per-service.html#comment-4076648710

Richardson, C. (2018d). *Event sourcing*. https://microservices.io/patterns/data/event-sourcing.html

Schaefer, M. (2020). *Onion Architecture explained — Building maintainable software | by Marco Schaefer | Medium*. https://marcoatschaefer.medium.com/onion-architecture-explained-building-maintainable-software-54996ff8e464

Schema Registry Overview | Confluent Documentation. (n.d.). Retrieved 28 May 2023, from https://docs.confluent.io/platform/current/schema-registry/index.html

Self-managed | SonarQube | Sonar. (n.d.). Retrieved 11 February 2023, from https://www.sonarsource.com/products/sonarqube/

Serilog | Sentry Documentation. (n.d.). Retrieved 9 June 2023, from https://docs.sentry.io/platforms/dotnet/guides/serilog/

Shemi, A. (2020). *Introduction to Test Driven Development in JS/Node.js, Part 1. | by akpojotor shemi | Medium*. https://medium.com/@pojotorshemi/introduction-to-test-driven-development-in-js-node-js-part-1-2477d9bbd3b5

Tapia, F., Mora, M. ángel, Fuertes, W., Aules, H., Flores, E., & Toulkeridis, T. (2020). From Monolithic Systems to Microservices: A Comparative Study of Performance. *Applied Sciences 2020, Vol. 10, Page 5797*, *10*(17), 5797. https://doi.org/10.3390/APP10175797

Testcontainers for .NET. (n.d.). Retrieved 14 May 2023, from https://dotnet.testcontainers.org/

Testing in .NET Core with xUnit and NSubstitute - Level Up Coding. (n.d.). Retrieved 17 June 2023, from https://levelup.gitconnected.com/testing-in-net-core-with-xunit-and-nsubstitute-a145c7b928c6

Unit testing C# code in .NET Core using dotnet test and xUnit - .NET | Microsoft Learn. (n.d.). Retrieved 14 May 2023, from https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test

Unit testing C# with NUnit and .NET Core (Testes de unidades em C# com NUnit e .NET Core) - .NET | Microsoft Learn. (n.d.). Retrieved 17 June 2023, from https://learn.microsoft.com/pt-pt/dotnet/core/testing/unit-testing-with-nunit

Unit Testing: Moq Framework | Microsoft Learn. (n.d.). Retrieved 14 May 2023, from https://learn.microsoft.com/en-us/shows/visual-studio-toolbox/unit-testing-moq-framework

UnitTest With AutoFixture In .NET 6.0. (2022). https://www.c-sharpcorner.com/blogs/unittest-with-autofixture-in-net-60

Welcome To Learn Dapper ORM - A Dapper Tutorial for C# and .NET Core. (n.d.). Retrieved 28 May 2023, from https://www.learndapper.com/

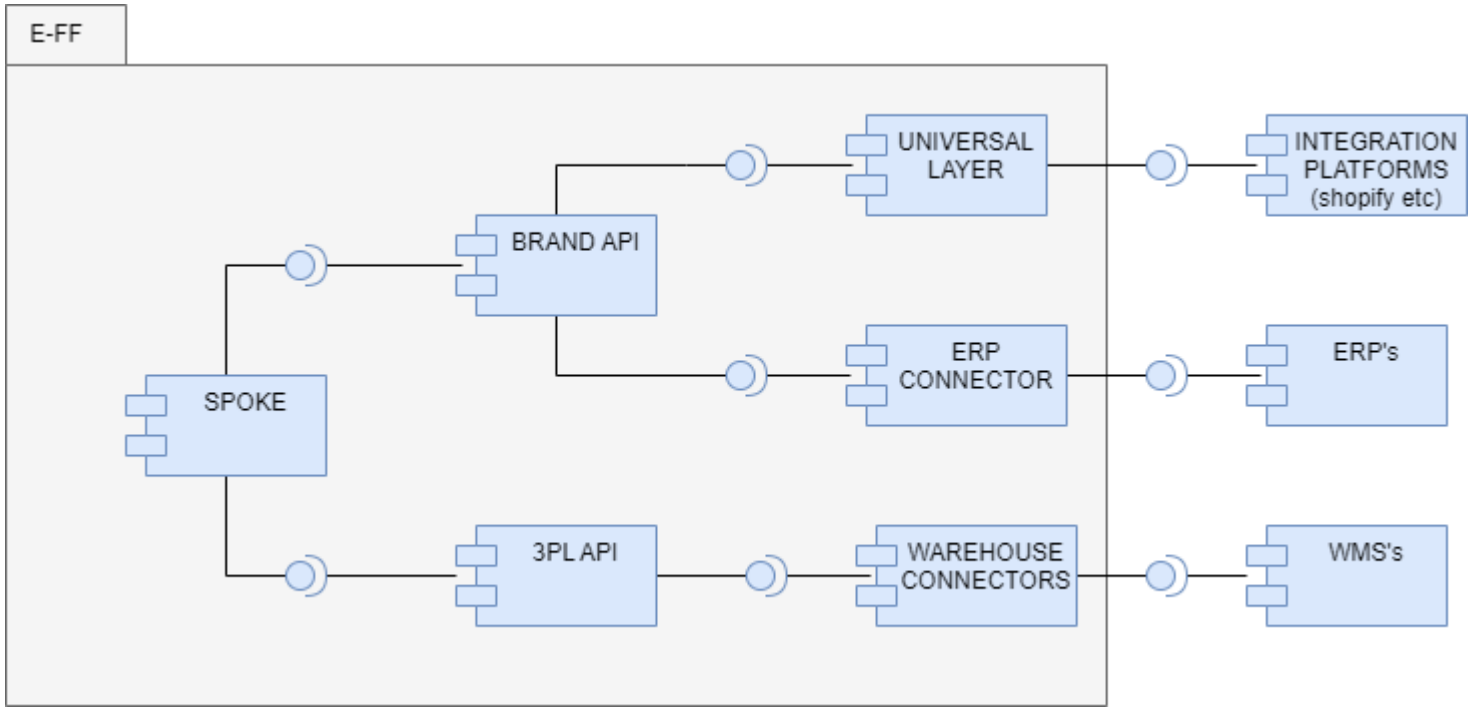
What Is Software ArchItecture: A Guide | Built In. (2022). https://builtin.com/software-engineering-perspectives/software-architecture

What is web2py. (n.d.). Retrieved 28 May 2023, from http://www.web2py.com/init/default/what

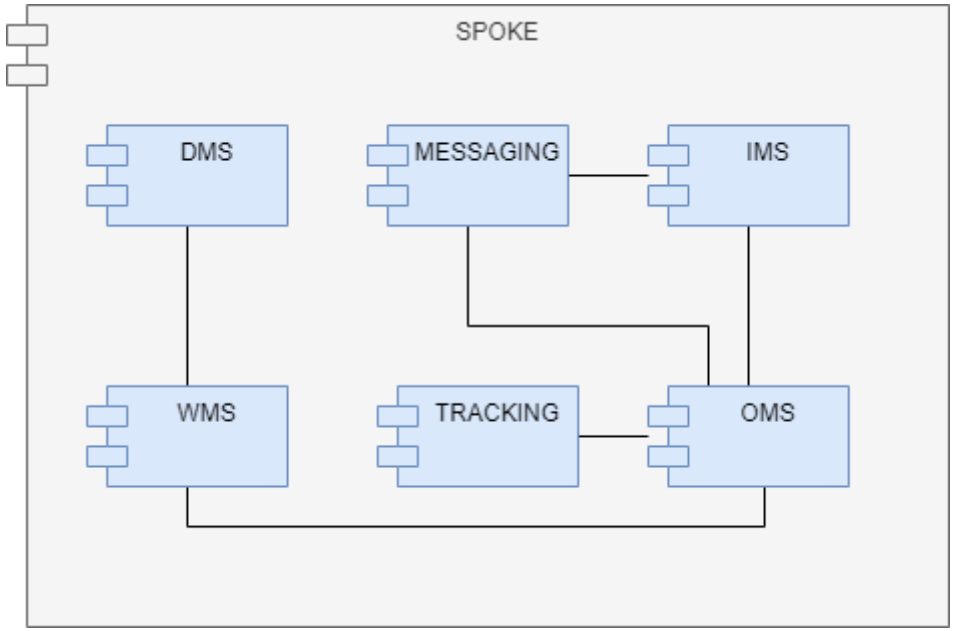
Wickramarathna, N. (2021). *Writing Better Performing Queries with LINQ on EF Core 6.0* . https://nishanc.medium.com/writing-better-performant-queries-with-linq-on-ef-core-6-0-%EF%B8%8F-85a1a406879

WireMock - flexible, open source API mocking | WireMock. (n.d.). Retrieved 14 May 2023, from https://wiremock.org/

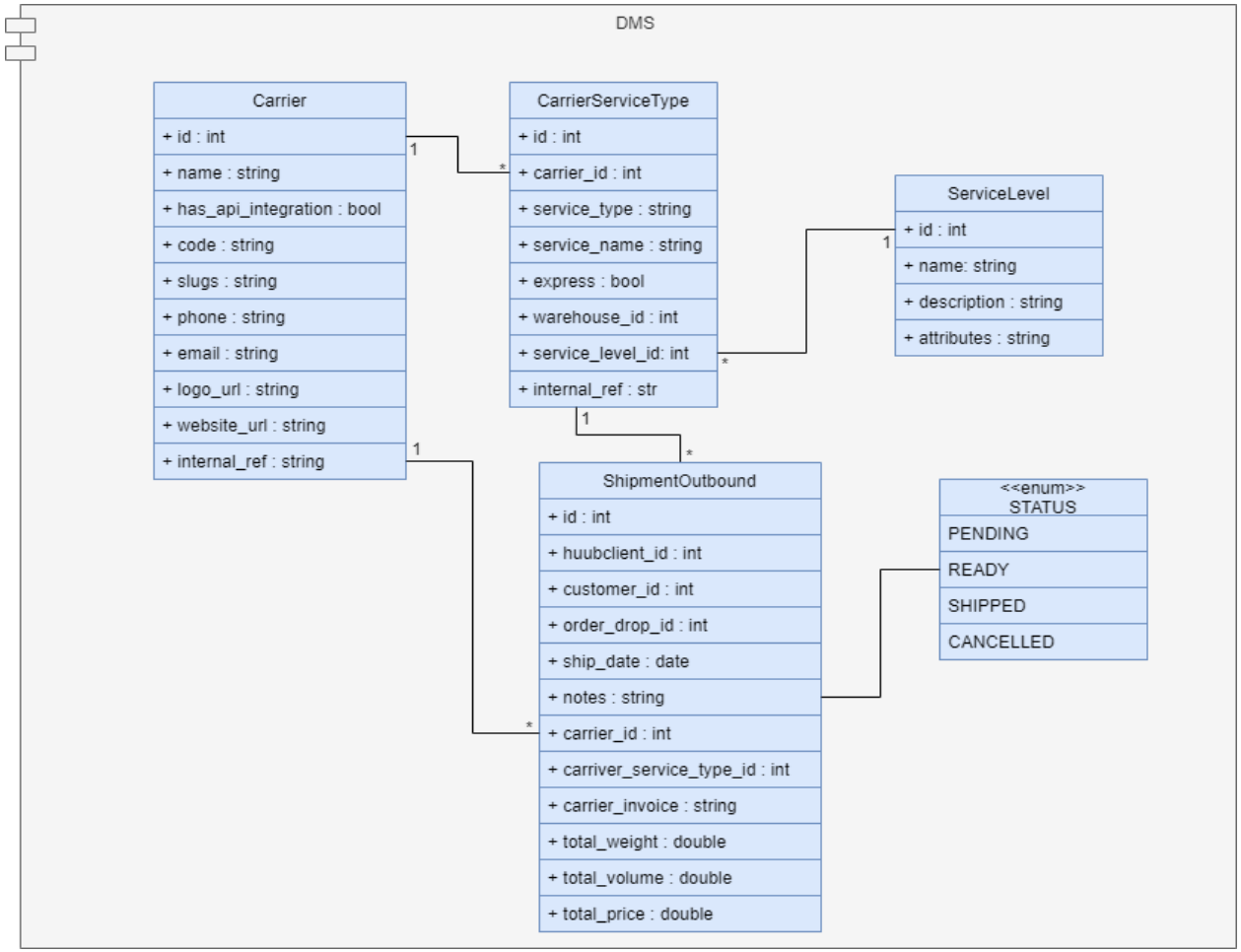
Anexo 1 – E-fulfillment Diagrama de componentes



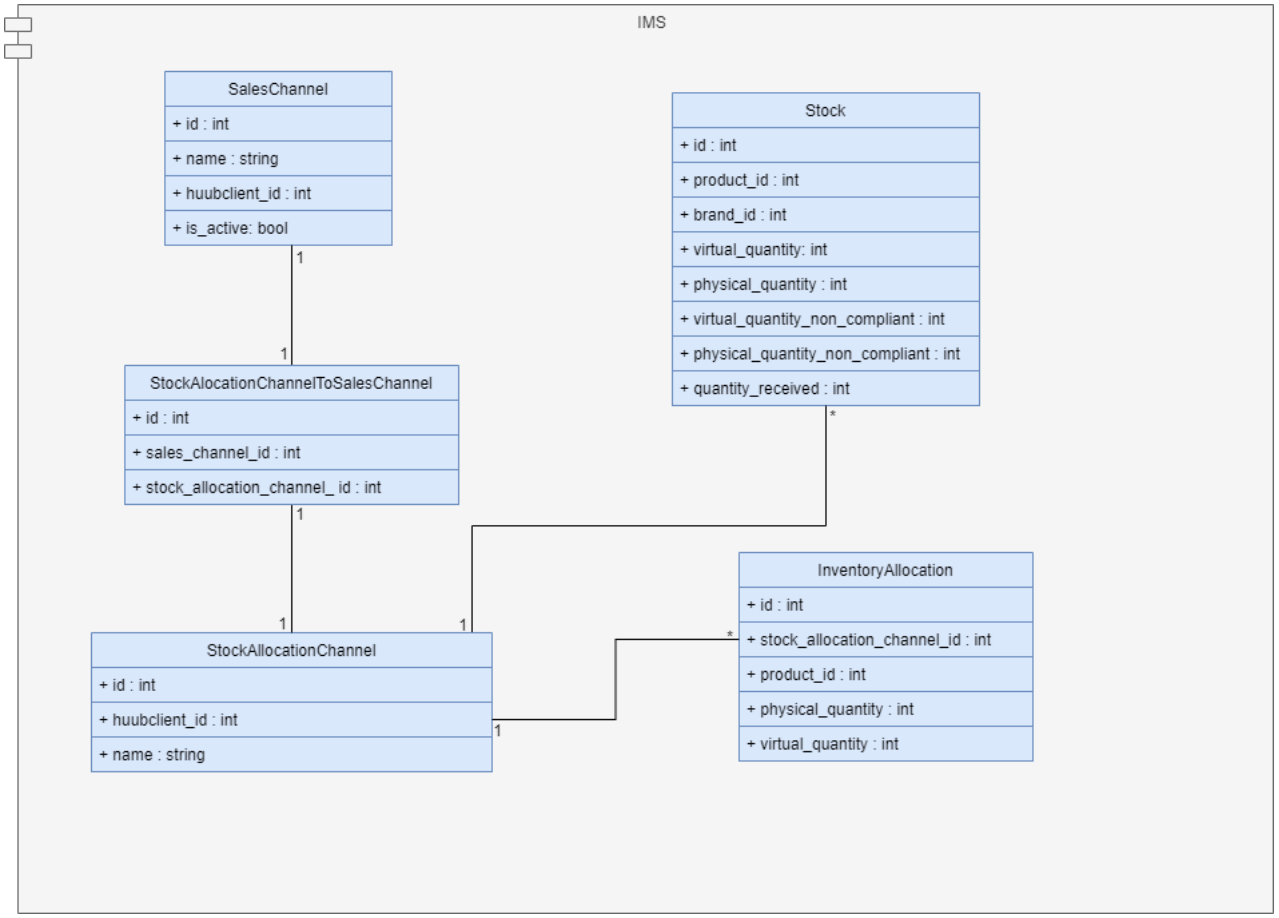
Anexo 2 – Spoke Diagrama de módulos



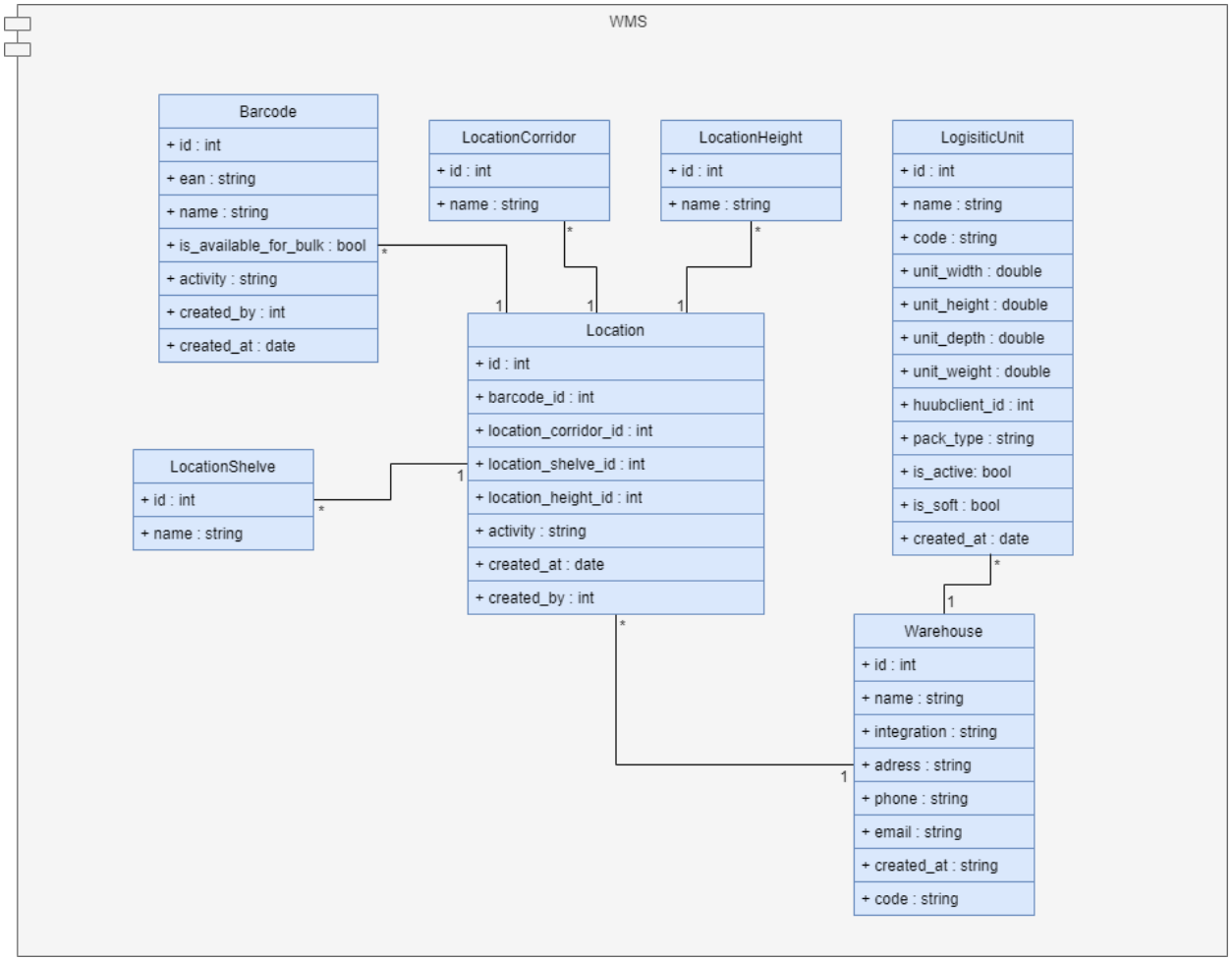
Anexo 3 – DMS Modelo de domínio



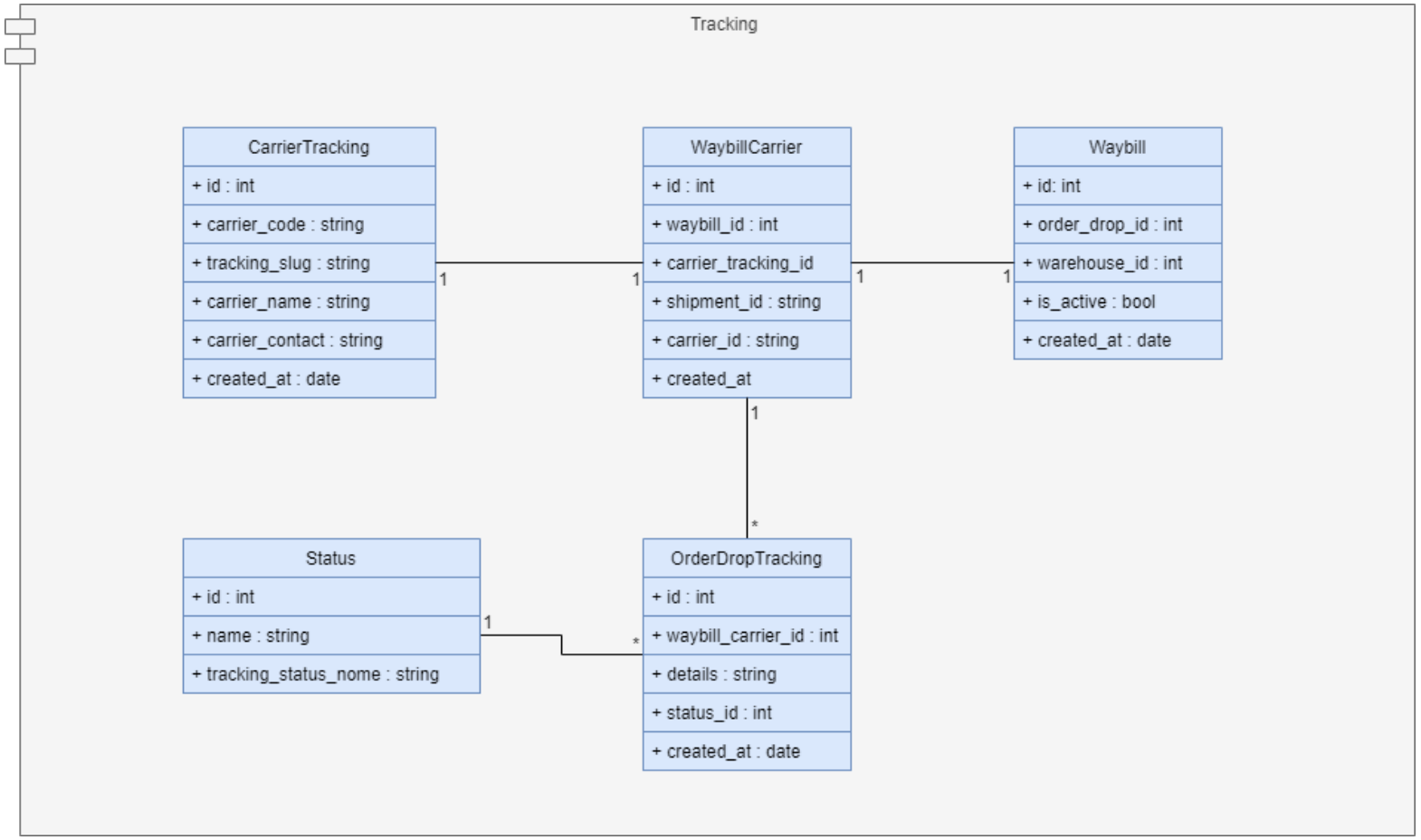
Anexo 4 – IMS Modelo de domínio



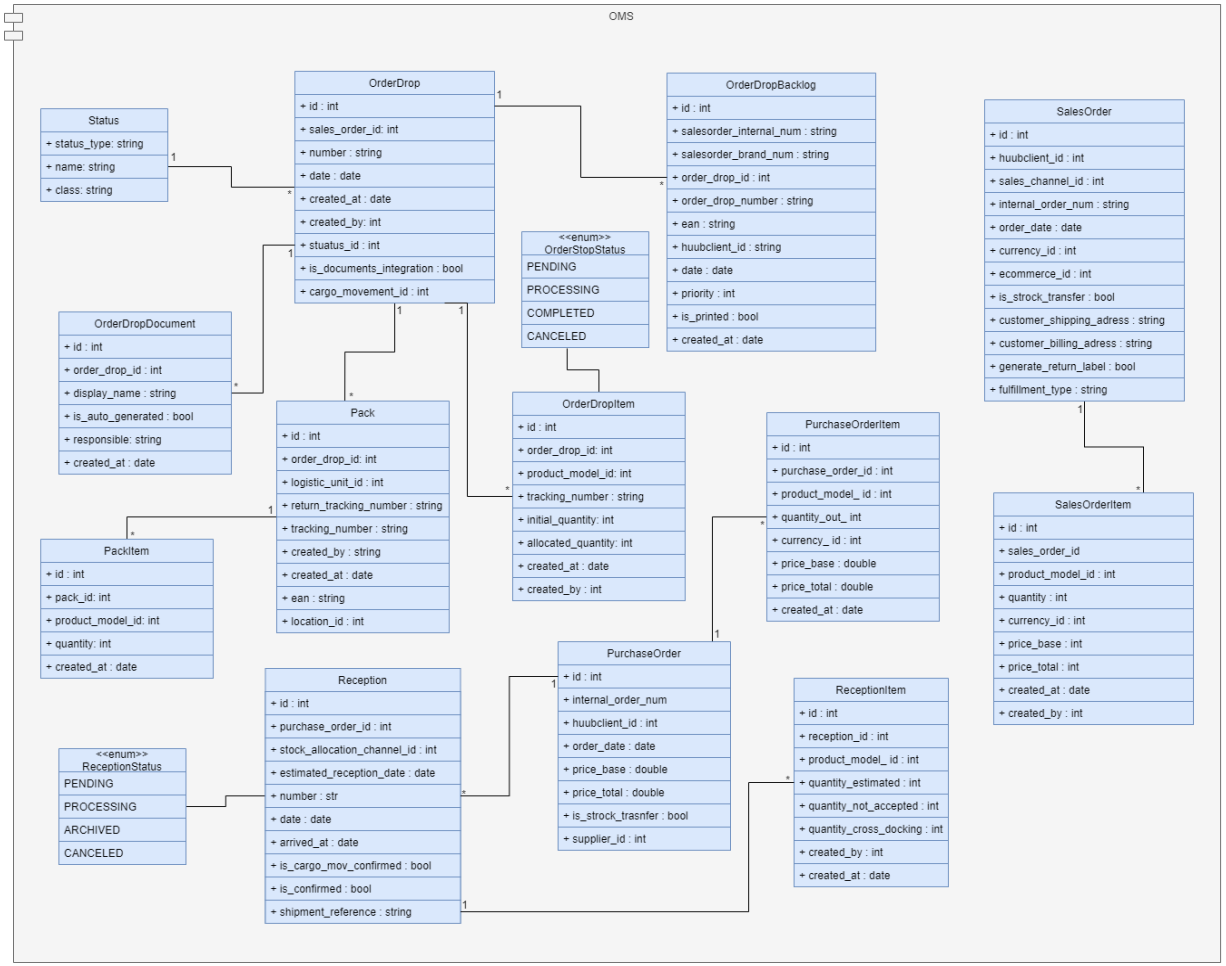
Anexo 5 – WMS Modelo de domínio



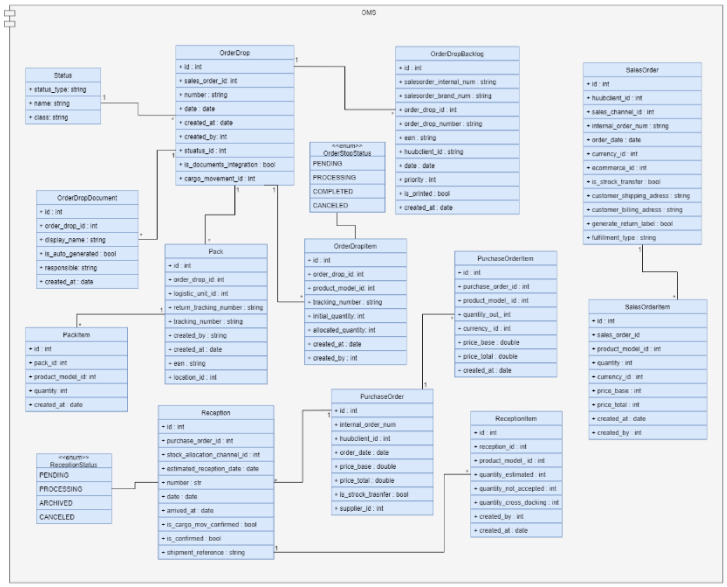
Anexo 6 – Tracking Modelo de domínio

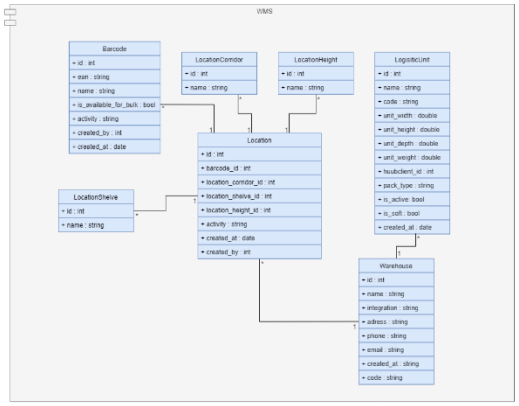


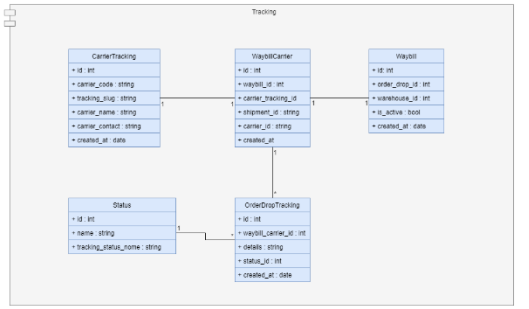
Anexo 7 – OMS Modelo de domínio

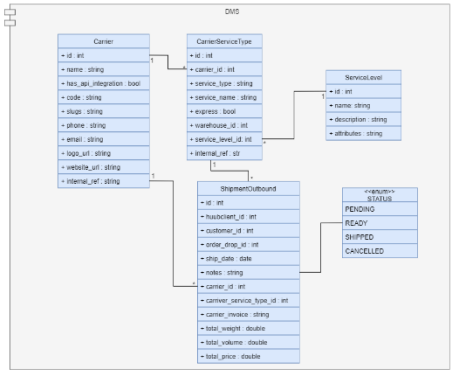


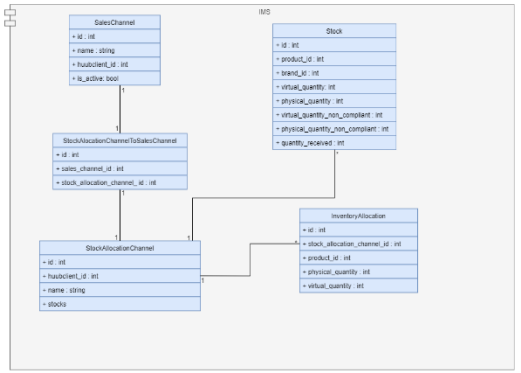
Anexo 8 – Primeira proposta arquitetural



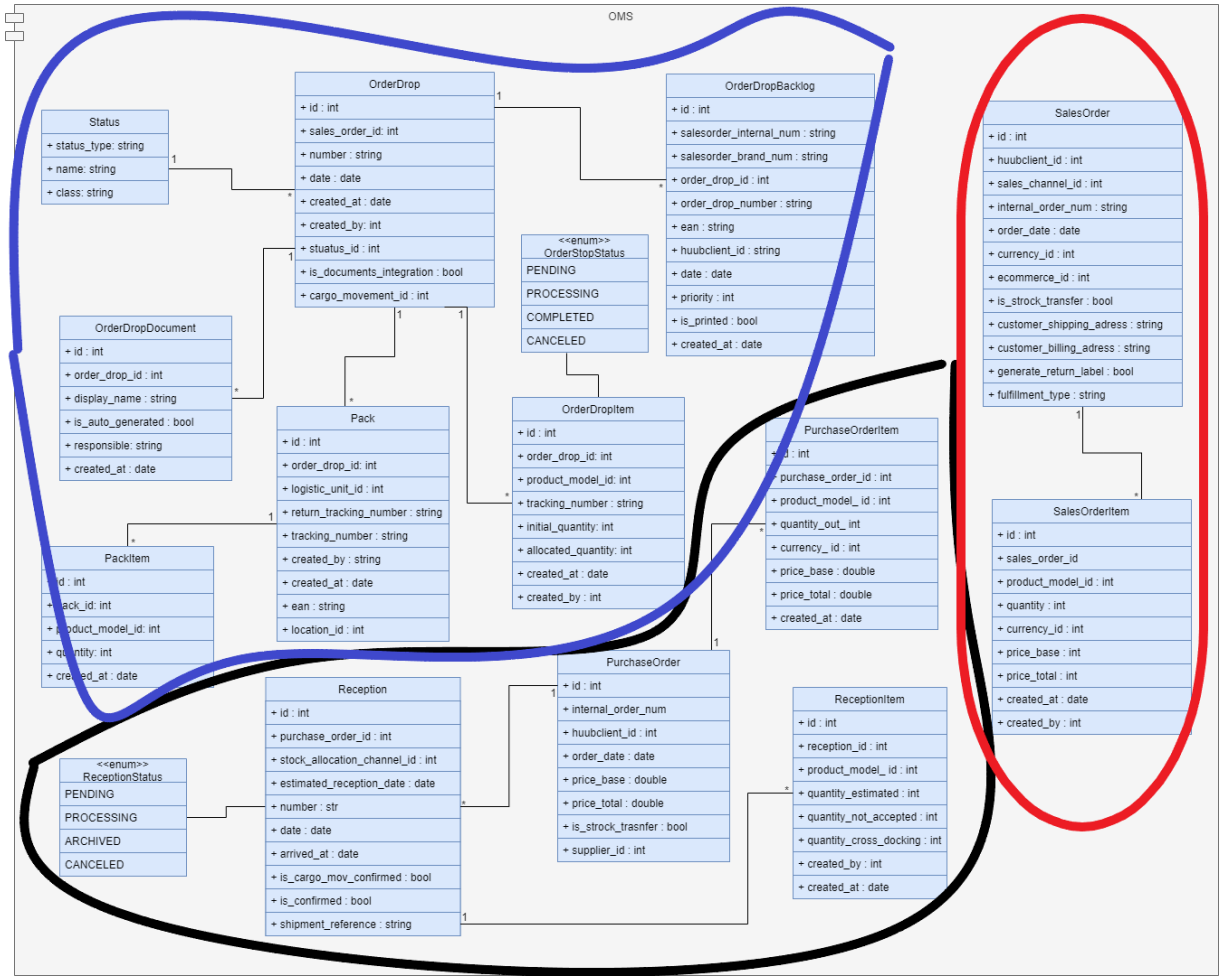






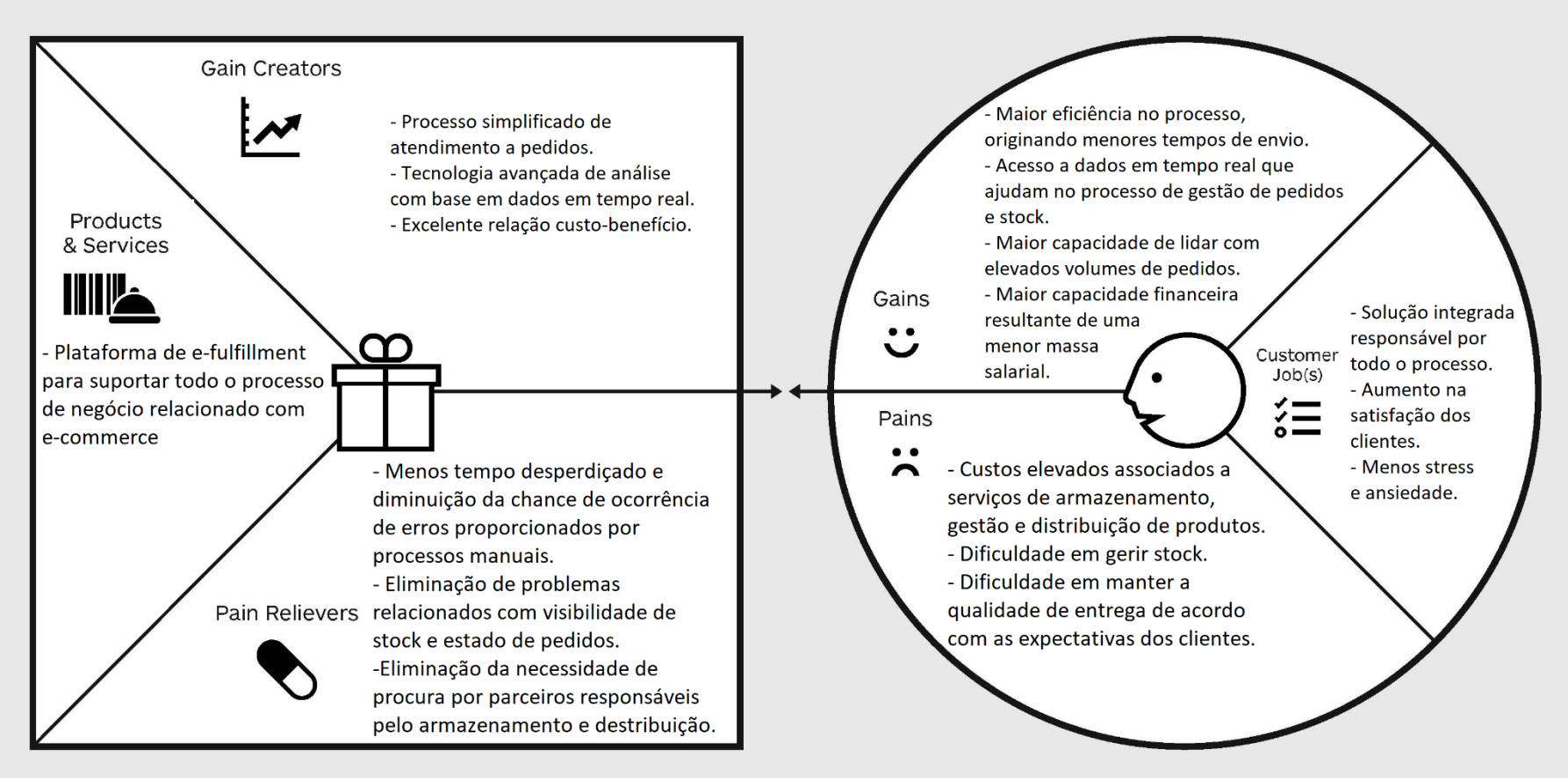


Anexo 9 – OMS Bounded Context



Anexo 10 – Segunda proposta arquitetural

Anexo 11 – Value proposition based on Osterwalder model



Anexo 12 – Questionário experimentação e avaliação

Microsserviços .NET para suportar uma plataforma de e-fulfillment

Este questionário, elaborado no âmbito da unidade curricular Tese/Dissertação/Estágio (TMDEI) do Mestrado em Engenharia de Software do Instituto Superior de Engenharia do Porto, tem como público alvo colaboradores que tomaram parte activa no desenvolvimento do projecto e têm como objectivo avaliar alguns dos objectivos definidos.

Secção 1

Perfil do público-alvo

1. Era colaborador da HUUB antes da compra por parte da Maersk? *

- ◯ Sim
- ◯ Não

2. Qual o cargo que melhor descreve o seu cargo atual? *

- ◯ Full Stack developer
- ◯ Backend developer
- ◯ Frontend developer
- ◯ Solution architect
- ◯ Team lead
- ◯ DevOps engineer
- ◯ Data engineer
- ◯ Product Owner

3. Quantos anos de experiência tem na área de desenvolvimento de software? *

- ◯ Menos de 2 anos
- ◯ Entre 2 e 5 anos
- ◯ Mais de 5 anos

4. Quantos anos de experiência possui nas tecnologias/*frameworks* utilizadas na Maersk? *

- ◯ Não possuo experiência
- ◯ Menos de 2
- ◯ Entre 2 e 5
- ◯ Mais de 5

5. Quantos anos de experiência tem em desenvolvimento de software tendo por base uma arquitectura orientada a microsserviços? *

- ◯ Não tenho experiência
- ◯ Menos de 2 anos
- ◯ Entre 2 e 5 anos
- ◯ Mais de 5 anos

6. Com base na sua experiência, como avalia a complexidade do desenvolvimento de sistemas com uma arquitectura orientada a microsserviços?

- ◯ 0
- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5

7. Alguma vez participou em um processo de migração arquitectural de um sistema monolítico para um sistema orientado a microsserviços? *

- ◯ Sim
- ◯ Não

8. Quais foram as principais dificuldades enfrentadas durante esse processo?

Selecione, no máximo, 3 opções.

- ☐ Identificação de "bounded contexts"
- ☐ Gestão da duplicação de informação entre os vários serviços
- ☐ Comunicação entre microsserviços
- ☐ Monitorização e "debugging"
- ☐ Organização entre equipas
- ☐ Testes e QA
- ☐ Deployments e infraestrutura
- ☐ Outro

Secção 2 ...

Interoperabilidade

Nesta secção pretende-se avaliar o grau de completude do objectivo referente à manutenção da antiga plataforma durante o processo de migração.

9. Face aos objectivos definidos, como avalia, de 0 a 5, sendo zero a pior avaliação possível, o grau de completude relativo à manutenção da plataforma antiga no decorrer do processo de migração? *

- ◯ 0
- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5

10. Como avalia, de 0 a 5, sendo zero correspondente a um nível de dificuldade baixo, a dificuldade associada à manutenção da plataforma antiga? *

- ◯ 0
- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5

11. Quais foram as principais dificuldades encontradas de forma a garantir a interoperabilidade entre plataformas? *

- ◯ Diferenças de conceitos
- ◯ Diferenças de processos
- ◯ Diferenças nas tecnologias utilizadas
- ◯ Limitações tecnológicas relativas à plataforma antiga
- ◯ Outro

12. Face aos objectivos definidos, como avalia, de 0 a 5, sendo que zero significa que a estratégia escolhida não foi a mais adequada, a estratégia escolhida para garantir a interoperabilidade entre plataformas? *

- ◯ 0
- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5

13. Face aos objectivos definidos, como avalia, de 0 a 5, sendo zero a pior avaliação possível, o grau de completude relativo ao desenvolvimento do projecto *template?* *

- ◯ 0
- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5

14. Quais das seguintes opções (pode seleccionar mais do que uma opção) melhor descrevem as principais vantagens em ter um projeto *template?* *

- ☐ Diminuição do tempo associado à criação de um novo serviço
- ☐ Diminuição do tempo associado à criação das pipelines de CI/CD para um novo serviço
- ☐ Uniformização de arquiteturas
- ☐ Uniformização de padrões
- ☐ Uniformização de bibliotecas
- ☐ Facilidade de manutenção

15. Com que frequência utiliza o projeto *template* aquando da criação de um novo serviço? *

- ◯ Não aplicável - Nunca criei um serviço
- ◯ Sempre
- ◯ Quase sempre
- ◯ Normalmente
- ◯ Quase nunca
- ◯ Nunca

16. Face aos objectivos definidos, como avalia, de 0 a 5, sendo zero a pior avaliação possível, a escolha arquitectural, padrões e bibliotecas escolhidas como padrão para o desenvolvimento dos novos microsserviços? *

- ◯ 0
- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5

Secção 4 ...

Análise arquitetural

Nesta secção pretende-se avaliar o grau de completude do objectivo referente à analise arquitectural efectuada e de que forma a arquitectura resultante dessa mesma análise ajudou no alcançar dos objectivos definidos.

17. Face aos objectivos definidos, como avalia, de 0 a 5, sendo zero a pior avaliação possível, de que forma a arquitectura desenvolvida contribuiu para a segregação de responsabilidades de domínio? *

- ◯ 0
- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5

18. Face aos objectivos definidos, como avalia, de 0 a 5, sendo zero a pior avaliação possível,, de que forma a arquitectura desenvolvida contribuiu para a segregação de responsabilidades entre equipas? *

- ◯ 0
- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5