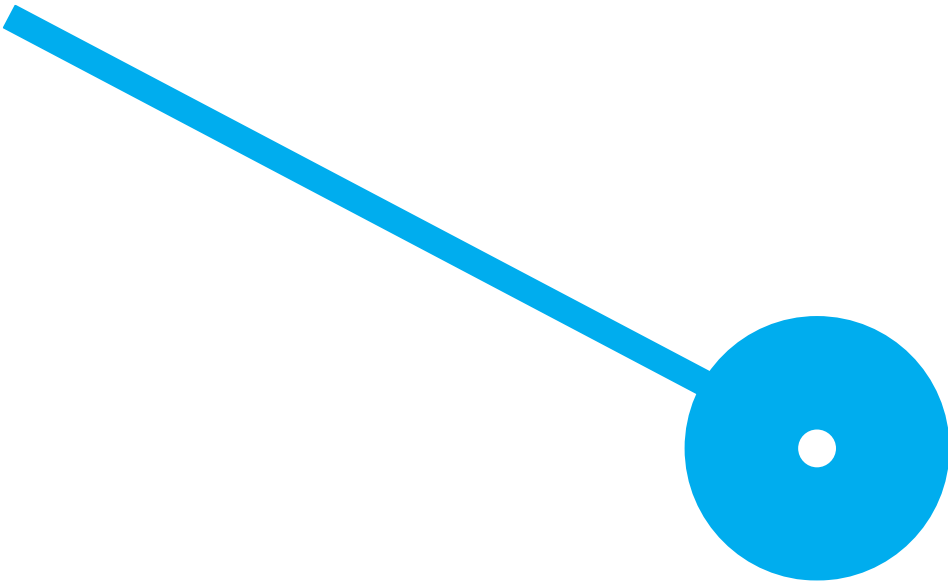
ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

MESTRADO
ENGENHARIA INFORMÁTICA

Solução Para Transformação Digital Orientada a Processos de Negócio

Pedro Manuel Pereira Afonso

12/2025



ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

MESTRADO
ENGENHARIA INFORMÁTICA

Solução Para Transformação Digital Orientada a Processos de Negócio

Pedro Manuel Pereira Afonso
8090457

Orientador

Professor Doutor João Paulo Ferreira de Magalhães

Relatório de Projeto apresentado para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática pela Escola Superior de Tecnologia e Gestão do Instituto Politécnico do Porto.

12/2025

Declaração de Integridade

Eu, Pedro Manuel Pereira Afonso, estudante nº 8090457, do Mestrado de Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico do Porto, declaro que não fiz plágio nem auto-plágio, pelo que o trabalho intitulado "Solução Para Transformação Digital Orientada a Processos de Negócio" é original e da minha autoria, não tendo sido usado previamente para qualquer outro fim. Mais declaro que todas as fontes usadas estão citadas, no texto e na bibliografia final, segundo as regras de referenciação adotadas na instituição.

Agradecimentos

Em primeiro lugar, quero expressar a minha profunda gratidão ao meu orientador, Professor Doutor João Paulo Magalhães, pela sua incansável disponibilidade, orientação e motivação contínua. O seu conhecimento, dedicação e incentivo foram essenciais para que este trabalho se concretizasse, inspirando-me a superar desafios e a crescer académica e pessoalmente.

Agradeço também à Escola Superior de Tecnologia e Gestão, assim como a todo o seu corpo docente e não docente, pela proximidade, apoio constante e dedicação ao longo de toda esta jornada. Foi essa dedicação, aliada à atenção e cuidado com cada estudante, que tornou este projeto possível.

À minha companheira de viagem, o meu sincero reconhecimento pelo seu apoio incondicional e por ser o meu porto seguro. A tua paciência, compreensão e encorajamento foram fundamentais para atravessar os momentos mais exigentes desta caminhada.

Aos meus filhos, pelo tempo que não estive presente, e a toda a minha família e amigos, agradeço pela compreensão, carinho e incentivo. Cada um de vós foi parte desta jornada e o vosso apoio constante tornou possível a concretização deste sonho.

Resumo

Os processos organizacionais manuais apresentam limitações de eficiência que comprometem a produtividade institucional. Este trabalho explora a aplicação de arquiteturas de microserviços na automação de processos de negócio incluindo o desenvolvimento de uma prova de conceito aplicada a um processo organizacional real para validar a viabilidade técnica desta abordagem.

O trabalho iniciou-se com a análise de diferentes paradigmas de arquitetura de software e a comparação de plataformas de automação de processos, nomeadamente Camunda, Bizagi e Bonita. O estudo comparativo conduziu à seleção do Camunda pela sua arquitetura distribuída, capacidade de integração com sistemas externos e compatibilidade com ambientes híbridos.

Após a escolha da plataforma, desenvolveu-se um estudo exploratório com o Camunda para validar as funcionalidades da plataforma. A solução desenvolvida baseia-se em microserviços containerizados orquestrados através de ambientes *Docker* e da distribuição *Kubernetes* denominada K3s. A automação de processos foi implementada utilizando a notação BPMN, inclui mecanismos de autenticação seguros e integração com a infraestrutura organizacional existente. A solução desenvolvida foi submetida de testes comparativos considerando diferentes configurações de orquestração. Os mesmos demonstraram compromissos entre performance, capacidades de gestão e disponibilidade. A monitorização contínua dos processos foi também tida em consideração na análise e a mesma permitiu analisar métricas operacionais e identificar padrões de utilização.

Em termos de contributo este trabalho produziu uma arquitetura que integra motores BPM e microserviços, padrões reutilizáveis de implementação e estratégias de containerização aplicáveis a diferentes ambientes. Os resultados validam a abordagem proposta como alternativa viável para projetos de digitalização. Conclui-se assim que este trabalho estabelece os fundamentos técnicos para implementações similares em organizações que procuram modernizar os seus processos.

Palavras-chave: Microserviços, BPM,BPMN, Camunda, *Docker*, K3s

Abstract

Manual organizational processes present efficiency limitations that compromise institutional productivity. This work explores the application of microservice architectures in business process automation, including the development of a proof of concept applied to a real organizational process to validate the technical feasibility of this approach.

The research began with an analysis of different software architecture paradigms and a comparison of business process automation platforms, namely Camunda, Bizagi, and Bonita. The comparative study led to the selection of Camunda due to its distributed architecture, integration capabilities with external systems, and compatibility with hybrid environments.

After selecting the platform, an exploratory study was conducted using Camunda to validate its functionalities. The developed solution is based on containerized microservices orchestrated through *Docker* environments and the K3s distribution of *Kubernetes*. Process automation was implemented using BPMN notation and includes secure authentication mechanisms as well as integration with the existing organizational infrastructure. The developed solution was subjected to comparative tests under different orchestration configurations, which revealed trade-offs between performance, management capabilities, and availability. Continuous process monitoring was also considered in the analysis, enabling the assessment of operational metrics and the identification of usage patterns.

In terms of contribution, this work produced an architecture that integrates BPM engines and microservices, reusable implementation patterns, and containerisation strategies applicable to different environments. The results validate the proposed approach as a viable alternative for digitalisation projects. It is therefore concluded that this work establishes the technical foundations for similar implementations in organisations seeking to modernise their processes.

Keywords: Microservices, BPM, BPMN, Camunda, *Docker*, K3s

Índice de Conteúdos:

Lista de Figuras:

Lista de Tabelas:

Lista de Siglas e Abreviaturas:

AMQP - Advanced Message Queuing Protocol

API - Application Programming Interface

AWS - Amazon Web Services

BCD - Bonita Continuous Delivery

BDM - Business Data Model

BPM - Business Process Management

BPMN - Business Process Model and Notation

BPMS - Business Process Management Suite

CI/CD - Continuous Integration/Continuous Deployment

CLI - Command Line Interface

CMIS - Content Management Interoperability Services

CNCF - Cloud Native Computing Foundation

CORS - Cross-Origin Resource Sharing

CPU - Central Processing Unit

CRUD - Create, Read, Update, Delete

CSS - Cascading Style Sheets

DDD - Domain-Driven Design

DMN - Decision Model and Notation

DNS - Domain Name System

DTO - Data Transfer Object

ECM - *Enterprise* Content Management

ELK - Elasticsearch, Logstash, and Kibana

ERP - *Enterprise* Resource Planning

ESB - *Enterprise* Service Bus

GB - Gigabyte

gRPC - Google Remote Procedure Call

HTML - HyperText Markup Language

HTTP - HyperText Transfer Protocol

HTTPS - HyperText Transfer Protocol Secure

IDE - Integrated Development Environment

IoT - Internet of Things

IP - Internet Protocol

JSON - JavaScript Object Notation

JWT - JSON Web Token

LDAP - Lightweight Directory Access Protocol

MB - Megabyte

MIME - Multipurpose Internet Mail Extensions

OAuth - Open *Authorization*

OData - Open Data Protocol

OMG - Object Management Group

ORM - Object-Relational Mapper

PDF - Portable Document Format

PFX - Personal Information Exchange

QA - Quality Assurance

QoS - Quality of Service

RAM - Random Access Memory

RBAC - Role-Based Access Control

REST - Representational State Transfer

SAFe - Scaled Agile *Framework*

SAML - Security Assertion Markup Language

SAP - Systems, Applications and Products in Data Processing

SDK - Software Development Kit

SLA - Service Level Agreement

SMTP - Simple Mail Transfer Protocol

SOA - Service-Oriented Architecture

SOAP - Simple Object Access Protocol

SQL - Structured Query Language

SPA - Single Page Application

SSO - Single Sign-On

TCP - Transmission Control Protocol

TI - Tecnologias da Informação

TLS - Transport Layer Security

UDP - User Datagram Protocol

URL - Uniform Resource Locator

UTC - Coordinated Universal Time

WSDL - Web Services Description Language

XML - eXtensible Markup Language

YAML - YAML Ain't Markup Language

1. Introdução

O presente capítulo apresenta o enquadramento do trabalho, define os objetivos do mesmo, e descreve a estrutura do documento. O estudo insere-se no contexto da transformação digital, explorando a aplicação de arquiteturas de microserviços e tecnologias de automação de processos de negócio em cenários organizacionais reais.

1.1. Enquadramento do Tema

A transformação digital tornou-se um imperativo estratégico para as organizações, que procuram modernizar os seus processos através da tecnologia. Esta necessidade é particularmente evidente em instituições onde procedimentos manuais, dependentes de documentação física e comunicação não estruturada, já não conseguem responder às exigências atuais de eficiência e transparência.

Este trabalho surge neste contexto de modernização organizacional, explorando como as arquiteturas de microserviços podem ser aplicadas na automação de processos de negócio. Para validar esta abordagem, foi selecionado um processo real que exemplifica os desafios típicos da transformação digital, nomeadamente: a existência de múltiplos intervenientes; a necessidade de aprovações hierárquicas (*workflow* de aprovação); a necessidade de gestão de recursos partilhados; a necessidade de rastreabilidade e monitorização. Este tipo de processo genérico, comum a diversas organizações, opera frequentemente através de formulários em papel, comunicações telefónicas e registos dispersos em folhas de cálculo, resultando em ineficiências operacionais significativas.

É importante referir que o trabalho não se limita à digitalização direta de procedimentos existentes. O mesmo aplica princípios de gestão de processos de negócio (BPM) e utiliza a notação Business Process Model and Notation (BPMN), desta forma é possível repensar o fluxo de trabalho para explorar as capacidades que a tecnologia oferece. A arquitetura de microserviços foi escolhida pela sua modularidade e escalabilidade, permitindo que diferentes componentes do sistema evoluam de forma independente. Um aspeto desta abordagem é a capacidade de integração com a infraestrutura existente nas organizações. A utilização de protocolos estabelecidos como LDAP para autenticação demonstra que é possível modernizar processos sem exigir a substituição completa dos sistemas em funcionamento. Esta característica é importante para viabilizar projetos de transformação digital em contextos reais, onde o investimento em infraestrutura *legacy* é significativo e a mudança deve ser gradual.

O caso de estudo implementado serve como prova de conceito, demonstra que os padrões e metodologias desenvolvidos podem ser generalizados para outros processos organizacionais com características similares, estabelecendo assim uma base técnica reutilizável para futuras iniciativas de digitalização.

1.2. Objetivos

O objetivo principal deste trabalho é analisar e validar a integração entre arquiteturas de microserviços e tecnologias de automação de processos de negócio, através de uma implementação prática que comprove a viabilidade técnica e os benefícios desta convergência na transformação digital organizacional.

Para alcançar este propósito, foram definidos objetivos específicos que orientaram o desenvolvimento do trabalho. Primeiramente, realizar uma análise comparativa dos paradigmas de arquitetura de software existentes, desde arquitetura monolítica, orientada a serviço e microserviços, com identificação das suas características, vantagens e limitações. Paralelamente, avaliar e comparar plataformas de gestão de processos de negócio disponíveis no mercado, estabelecendo critérios de seleção baseados em capacidades técnicas e escalabilidade.

A componente prática do trabalho visa implementar uma solução completa que materialize os conceitos estudados. Isto inclui o desenvolvimento de microserviços especializados com padrões de comunicação, segurança e gestão de dados em ambientes distribuídos. A integração com motores de orquestração de processos constitui outro objetivo, onde será explorado como estas tecnologias podem coexistir e complementar-se na automação de *workflows*. Ainda no âmbito da componente prática é objetivo deste trabalho, validar experimentalmente a solução através de testes comparativos entre ambientes *Docker* e a distribuição *Kubernetes* denominada K3s, para avaliar os compromissos entre performance e capacidades de orquestração. Outro objetivo é o de monitorizar e analisar métricas de desempenho que inclui tempos de resposta, consumo de recursos e disponibilidade do sistema, de forma a aferir otimizações suportadas em evidências.

De forma geral, este trabalho também visa estabelecer uma base metodológica que oriente futuras iniciativas de transformação digital em organizações com requisitos similares.

1.3. Contribuições Esperadas

Com a execução deste trabalho prevê-se atingir resultados que validam a abordagem proposta e disponibilizar contribuições técnicas que avançam o conhecimento na área de integração entre arquiteturas de microserviços e automação de processos.

Do ponto de vista da implementação prática, pretende-se o desenvolvimento de uma solução completa que digitaliza um processo organizacional real. Deste modo evidencia-se a forma como um procedimento inteiramente manual poderá ser transformado para tirar proveito de um *workflow* automatizado, evidenciando as vantagens. O trabalho pretende ainda mostrar a viabilidade da integração entre motores de processos e arquiteturas baseadas em microserviços, traduzindo-se na segmentação das soluções em microserviços e na sua visão integrada através dos processos de negócio.

Outro contributo importante para a comunidade deriva dos testes comparativos entre *Docker* e K3s. Estes testes visam avaliar o comportamento do sistema em diferentes configurações, incluindo execução em container único e distribuição por um e múltiplos nodes. Os resultados dos testes servem de base às escolhas no âmbito deste trabalho e também de trabalhos futuros, incluindo o de outros profissionais ou estudantes da área.

Um outro aspeto a destacar neste projeto refere-se à interoperabilidade entre os sistemas, evidenciando que o processo de modernização não implica necessariamente a substituição das infraestruturas tecnológicas existentes.

Em suma, como contribuições, o trabalho produzirá uma metodologia estruturada para integração entre motores BPM e microserviços. Esta metodologia documenta desde a identificação de contextos delimitados até à implementação de comunicação assíncrona, incluindo questões como tolerância a falhas, monitorização dos serviços e gestão de autenticação. A viabilidade de migração progressiva de microserviços desde ambiente nativo para containers e a posterior orquestração servirá também de base aos futuros projetos. O papel do sistema de monitorização capaz de transformar dados operacionais em informação acionável contribui para a tomada de decisão baseada em fatos. Os alertas automáticos a definir para identificar situações anómalas, como tempos de resposta superiores aos limites estabelecidos são uma mais-valia no âmbito deste tipo de projetos onde se procura uma otimização global dos processos de negócio. A capacidade analítica a introduzir define uma abordagem

preventiva suportada por monitorização em tempo real bem como facilitar a mudança de uma gestão reativa do processo manual por uma gestão informada e proativa.

1.4. Estrutura da dissertação

A dissertação encontra-se organizada em seis capítulos que estruturam o projeto de forma lógica e sequencial, desde o enquadramento teórico até à validação experimental da solução proposta.

O Capítulo 1 contextualiza a investigação no domínio da transformação digital organizacional, apresenta os objetivos do trabalho e sintetiza os principais resultados a atingir. Define ainda o âmbito da investigação e os aspetos centrais analisados no estudo da integração entre arquiteturas de microserviços e plataformas de gestão de processos de negócio.

O Capítulo 2 desenvolve o enquadramento teórico necessário à compreensão dos domínios tecnológicos abordados. A revisão bibliográfica abrange estratégias de transformação digital, fundamentos da gestão de processos de negócio, análise comparativa de arquiteturas de software desde monolíticas até microserviços, e avaliação de plataformas de automação de processos. Inclui casos de estudo reais de digitalização organizacional e análise comparativa das plataformas Camunda, Bizagi e Bonita BPM.

O Capítulo 3 documenta a fase inicial de validação através do desenvolvimento de uma prova de conceito baseada na plataforma Camunda. Apresenta o levantamento do processo de requerimentos da Escola Superior de Tecnologia e Gestão (ESTG), a implementação de um modelo BPMN funcional e a análise das capacidades e limitações identificadas durante os testes exploratórios.

O Capítulo 4 detalha a arquitetura proposta e a metodologia de desenvolvimento adotados. Aborda os padrões de microserviços implementados, desde *Domain-Driven Design* simplificado até *Circuit Breaker Pattern*, apresenta as tecnologias selecionadas para *backend*, *frontend*, infraestrutura e automação, e documenta o levantamento do processo de autorização de condução de viatura. Inclui a definição dos *bounded contexts* e o mapeamento para microserviços.

O Capítulo 5 apresenta a implementação técnica da solução através da descrição dos microserviços desenvolvidos, padrões de comunicação e integração. Documenta a migração de ambiente nativo para *Docker* e posterior migração para K3s, a implementação do *frontend* em Angular, o sistema de monitorização com *Grafana*, os testes implementados, *logging* e documentação das API.

O Capítulo 6 apresenta as conclusões do trabalho, os principais contributos e identifica limitações do estudo bem como propostas para trabalho futuro.

Esta estrutura assegura uma progressão lógica desde os fundamentos teóricos até à implementação prática, permitindo a replicação da metodologia proposta em contextos organizacionais similares.

2. Estado da Arte

Este capítulo desenvolve a fundamentação teórica necessária à compreensão dos domínios tecnológicos abordados neste trabalho. A revisão da literatura estrutura-se em quatro áreas principais, incluindo estratégias para a transformação digital organizacional, modelação e gestão de processos na transformação digital, análise comparativa de arquiteturas de software desde monolíticas até microserviços, e avaliação de plataformas de automatização de processos. São apresentados casos reais de digitalização organizacional em diferentes setores e desenvolvida uma análise comparativa das plataformas Camunda, Bizagi e Bonita BPM, fundamentando as escolhas tecnológicas subsequentes.

2.1. Estratégias para a Transformação Digital

Atualmente, assistimos a uma crescente pressão sobre as organizações para adotarem práticas e tecnologias que promovam a sua adaptação ao mundo digital. Neste contexto, a transformação digital é analisada não apenas como uma tendência tecnológica, mas como um processo estratégico que envolve mudanças estruturais, organizacionais e de modelo de negócio. Neste sentido, importa compreender as estratégias associadas à transformação digital e os fatores que contribuem para a sua implementação eficaz.

De acordo com o referido em [1], a transformação digital tornou-se uma força dominante em todas as áreas, causando mudanças acentuadas na forma como as organizações funcionam e se posicionam no mercado. As estratégias de transformação digital apresentam características que podem ser divididas em quatro dimensões: utilização de tecnologias; mudanças na criação de valor; ajustes estruturais e ajustes financeiros. Ao abordar o uso de tecnologia, a empresa define o seu papel estratégico em relação às inovações tecnológicas a introduzir. As mudanças na criação de valor influenciam a necessidade de adaptações na estrutura organizacional, enquanto as financeiras desempenham um papel fundamental afetando a capacidade de financiamento para a transformação digital bem como num possível retorno dessa transformação. Os autores em [2] aprofundam esta temática sendo referido o papel dos requisitos funcionais e não funcionais e características técnicas, nomeadamente métricas de desempenho como latência e propriedades como segurança para a correta transformação digital. Os autores no trabalho apresentado em [3] defendem que a definição detalhada das características do negócio é fundamental para o sucesso da transformação digital. Neste trabalho são identificadas quatro etapas para a transformação digital: seleção e modelação do processo de negócio,

avaliação das tecnologias digitais com foco no comportamento do processo, inclusão de perspetivas adicionais (outros aspetos fundamentais, objetivos e riscos) e escolha das tecnologias mais adequadas. Embora sejam conhecidas histórias de sucesso decorrentes de uma abordagem de impulso tecnológico, é crucial ponderar os custos e benefícios associados. Tal como referido em [4], a implementação da tecnologia, se não for realizada conforme o planeado, pode representar um risco para a organização. Isto reforça a importância de seguir uma abordagem adequada à transformação digital, colocando o negócio em primeiro e a transformação digital do mesmo numa perspetiva tecnológica como motor de impulso e inovação. Os autores na publicação [5] referem os desafios inerentes à estratégia de transformação digital, reforçando que uma gestão estratégica eficiente e uma avaliação precisa das necessidades de digitalização da organização, deve ser tida em consideração para maximizar o retorno da transformação digital.

2.2. Modelação e Gestão de Processos na Transformação Digital

Após a definição de estratégias para a transformação digital, é relevante compreender como os processos de negócio devem ser estruturados e modelados para suportar essa transformação. Neste contexto, a gestão e a modelação de processos de negócio desempenham um papel central.

2.2.1. Enquadramento da Gestão de Processos de Negócio

A Gestão de Processos de Negócios emerge como um campo dinâmico, cuja definição não se limita a algo específico. Ao longo das décadas, a BPM evoluiu significativamente, tornando-se a ciência e prática de supervisão de execução do trabalho, com o objetivo intrínseco de garantir resultados consistentes e identificar oportunidades para melhorias. No contexto dessa evolução procura-se explorar as principais linhas da BPM, com ênfase na modelação de processos de negócios.

O trabalho apresentado em [3], estabelece que a essência da BPM reside em métodos, técnicas e ferramentas que propiciam a melhoria, execução, gestão e análise dos processos organizacionais. Destaca a importância da melhoria contínua, caracterizada como um processo metódico de avaliação, análise e melhoria dos processos fundamentais para o sucesso organizacional. Por sua vez, a modelação de processos de negócios, conforme proposto, pelos autores em [6] revela-se como uma ferramenta padronizada, essencial para a compreensão abrangente dos elementos que compõem os processos, tais como atividades, participantes, mecanismos de controlo, recursos envolvidos e saídas. A modelação organizacional procura atingir propósitos de alto nível, desde a compreensão até a

regulamentação dos processos. No entanto, Jeston e Nelis em [7] destacam a contínua relevância do BPM, evidenciando a constante procura por melhorar os processos de negócios, sublinhando o papel do BPM como uma filosofia de gestão. Esta orientação a processos visa a organização de pessoas para alcançar maior agilidade nos ambientes empresariais.

2.2.2. O Ciclo de Vida da Gestão de Processos de Negócio

Quando uma organização dá início à digitalização dos seus processos, está a começar o ciclo de vida da Gestão dos Processos de Negócio, desencadeando o ponto de partida para a estruturação e melhoria constante dos processos. Após analisar várias fontes, fica evidente que diferentes autores apresentam variações nos ciclos de vida BPM. Segundo o apresentado em [8], o ciclo de vida da BPM não é apenas uma sequência linear, mas sim um processo contínuo e interativo, destacando-se pela constante revisão e continuidade. De acordo com o estudo, o ciclo é constituído por quatro fases:

1. **Fase de Design:** Redefinição dos processos.
2. **Fase de Configuração:** Implementação dos projetos através da configuração de um sistema de informação voltado para os processos.
3. **Fase de Execução:** Início da execução dos processos operacionais utilizando o sistema de informação.
4. **Fase de Diagnóstico:** Análise dos processos operacionais para identificar problemas e possíveis melhorias.

No entanto, de acordo com o apresentado em [3], o ciclo de vida BPM envolve diversas atividades cruciais, desde a identificação até a melhoria dos processos. Estas atividades incluem etapas como identificação, definição, modelação, implementação, execução, monitorização, controlo e melhoria dos processos. O estudo apresentado em [9] enfatiza também o design, modelação, execução, monitorização e otimização. Cada fase desempenha um papel singular e fundamental na estruturação, implementação e contínuo refinamento dos processos organizacionais, com o objetivo contínuo de melhorar a eficiência. Além disso, conforme proposto no estudo "*Fundamentals of Business Process Management*" em [10], complementa-se estas abordagens, enfatizando a importância da identificação, descoberta, análise, redesenho, implementação, monitorização e controlo, que, em conjunto, formam um quadro cíclico e iterativo para a melhoria contínua dos processos. A identificação permite reconhecer os processos críticos para a organização, servindo de base para a seleção daqueles que necessitam de atenção. A fase de descoberta foca-se na modelação do processo tal como é, recorre a técnicas como entrevistas, *workshops,* análise de documentação para representar e perceber o funcionamento atual. Segue-se a análise, onde são identificadas ineficiências, gargalos ou riscos, que

fundamentam o redesenho do processo, fase que propõe alterações orientadas para a otimização de desempenho. A implementação traduz as melhorias em mudanças práticas, através de tecnologias, novos papéis organizacionais ou sistemas de informação. Finalmente, as fases de monitorização e controlo asseguram a avaliação contínua do desempenho do processo, permitindo ajustes ao longo do tempo. Ao integrar estas etapas de forma coerente, o modelo de Dumas proporciona uma abordagem sistemática que reforça a maturidade da gestão de processos e sustenta a transformação digital das organizações com base em evidências e melhoria incremental [10] .

2.2.3. Business Process Model and Notation (BPMN)

A BPMN, tal como apresentado em [11], é uma notação gráfica padronizada desenvolvida pela *Object Management Group* (OMG), com o objetivo de oferecer uma linguagem comum para descrever processos de negócio de forma clara tanto para analistas de negócio como para técnicos responsáveis pela implementação de soluções. A BPMN permite representar processos, como ilustrado na Figura 1 através de diagramas que articulam diferentes tipos de elementos, organizados em quatro categorias principais: *Flow Objects* (eventos, atividades e *gateways*), *Connecting Objects* (sequência, mensagem e associação), *Swimlanes* (*pools e lanes*) e *Artifacts* (dados, grupos e anotações). Esta estrutura fornece uma base para modelar tanto processos internos quanto colaborações interorganizacionais, suportando níveis crescentes de complexidade.

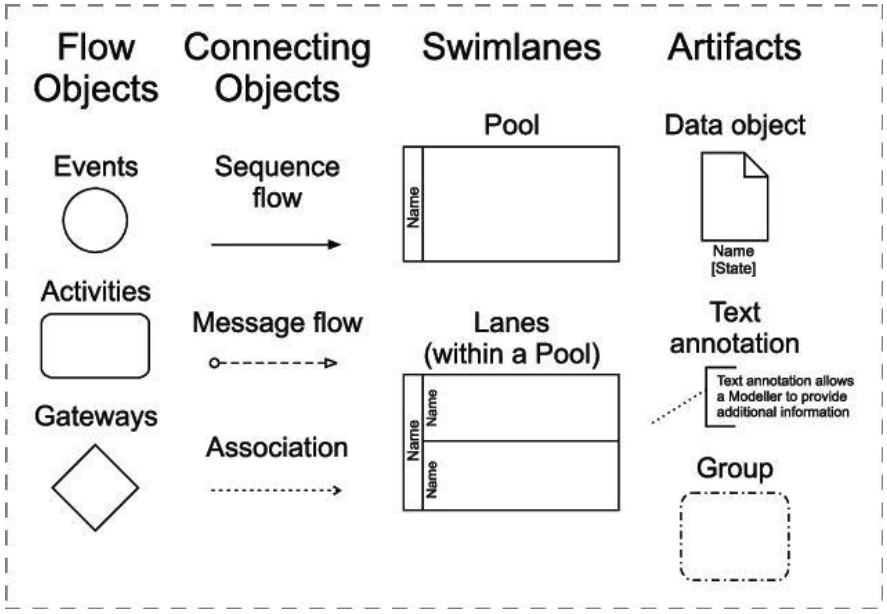


Figura 1 - Representação gráfica dos elementos BPMN, adaptado de [11]

A utilização de BPMN tem sido particularmente relevante em iniciativas de Business Process Management, dada a sua capacidade de ligar a modelação conceptual com a execução técnica. Segundo os autores em [13], a BPMN assume um papel relevante na transição entre o design conceptual e a implementação prática dos processos de negócio. Os autores indicam que os diagramas BPMN podem ser integrados em sistemas BPMS (*Business Process Management Suites*) ou motores de *workflow*, possibilitando a execução automática dos processos. Além disso, demonstram como a orquestração com dispositivos *Internet of Things* (IoT) pode ser realizada através de uma arquitetura baseada em microserviços, promovendo uma automatização distribuída, capaz de responder a contextos dinâmicos e ambientes físicos interligados. Neste contexto, os autores em [14] conduziram um estudo baseado em *eye-tracking* com 21 profissionais de software que demonstrou que os *horizontal layouts sem scroll* são mais fáceis de ler, exigindo menor tempo de resposta e menor carga cognitiva. Quando o diagrama ultrapassava a área visível, *layouts* em *snake* ou multi-linha eram preferíveis ao *scroll* horizontal e vertical, pois melhoravam significativamente a legibilidade e compreensão do conteúdo. Assim, as orientações práticas recomendadas incluem:

- Utilização prioritária de *layout* horizontal;
- Evitar *scroll* horizontal ou vertical durante a leitura do diagrama;
- Empregar *layouts* em *snake* ou multi-linha quando o diagrama for extenso, evitando fragmentar a visualização.

As práticas apresentadas estão alinhadas com conceitos reconhecidos em *guidelines* de modelação, reforçando a ideia de que a estrutura visual do processo tem impacto direto na sua compreensão. Embora a BPMN ofereça uma linguagem muito usada em ambientes empresariais para representar e comunicar processos de negócio, continua a apresentar desafios no que diz respeito à verificação formal e à garantia da correção dos modelos. Segundo o que o autor apresenta em [12], a BPMN, por si só, não fornece suporte direto à análise de propriedades formais, como *deadlocks*, *reachability* ou comportamento temporal, o que limita a sua aplicabilidade em contextos que exigem validação de requisitos e conformidade operacional. O autor propõe, como resposta a esta limitação, a integração da BPMN com técnicas de *model checking*, o que permite que os modelos desenhados em BPMN possam ser convertidos para linguagens formais e analisados computacionalmente. Esta abordagem oferece uma via para mitigar as fragilidades da notação em ambientes críticos, onde a fiabilidade dos processos é fundamental. Este é um exemplo dos avanços que reforçam a sua utilidade na transformação digital e na melhoria contínua de processos de negócio.

2.3. Arquiteturas de Software Monolítica, SOA e de Microserviços e sua Comparação

A definição da arquitetura de software constitui uma decisão estratégica que influencia a escalabilidade, a manutenção, a flexibilidade e a eficiência dos sistemas de informação. Ao longo das últimas décadas, três abordagens de arquitetura destacaram-se pela sua adoção e evolução conceptual, nomeadamente a arquitetura monolítica, a arquitetura orientada a serviços (SOA) e a arquitetura baseada em microserviços. Cada uma destas representa um paradigma com pressupostos técnicos, operacionais e organizacionais distintos, adaptando-se a diferentes contextos de aplicação e maturidade tecnológica. Este subcapítulo propõe uma análise comparativa entre estas três arquiteturas, com base em critérios como modularidade, dependência entre componentes, escalabilidade, comunicação e complexidade operacional, com o objetivo de apoiar a escolha fundamentada da abordagem mais adequada a diferentes cenários de desenvolvimento e implementação de sistemas.

2.3.1. Arquitetura Monolítica

A arquitetura monolítica, conforme descrito em [15], representa um modelo clássico de desenvolvimento de sistemas de software, no qual todos os componentes de uma aplicação, incluindo a interface com o utilizador, a lógica de negócio e o acesso a dados, são implementados e executados como uma única unidade de implantação. Esta abordagem tem sido historicamente predominante devido à sua simplicidade de estrutura, facilidade de desenvolvimento inicial e reduzida complexidade operacional. Contudo, com a crescente popularização de arquiteturas distribuídas, como microserviços e arquiteturas baseadas em funções, a arquitetura monolítica tem sido alvo de um renovado escrutínio no seio da engenharia de software moderna. Ainda assim, estudos recentes confirmam a sua pertinência em determinados contextos, nomeadamente em sistemas de pequena ou média escala, em ambientes com equipas reduzidas ou em cenários com baixos requisitos de escalabilidade. De acordo com o trabalho apresentado em [16], o modelo monolítico baseia-se na premissa de que todos os módulos da aplicação partilham o mesmo espaço de memória e são executados no mesmo processo. A gestão do ciclo de vida da aplicação realiza-se de forma conjunta, implicando que qualquer alteração no código, ainda que localizada, exige a recompilação e reimplantação de toda a aplicação. Esta característica, embora eficiente em contextos controlados, pode tornar-se um fator limitador à medida que o sistema cresce em complexidade funcional. Apesar disso, uma das principais vantagens associadas a esta abordagem reside na eficiência da comunicação interna entre os componentes, uma vez que esta ocorre *in-memory* e não depende de chamadas

remotas, como sucede em arquiteturas baseadas em serviços. Além disso, em [17] é referido que a arquitetura monolítica permite um processo de desenvolvimento mais direto e coeso, reduzindo o tempo necessário para a configuração de ambientes e a integração de múltiplos serviços. A gestão centralizada da aplicação facilita também a monitorização do sistema, a deteção de falhas e a aplicação de correções, uma vez que todos os módulos estão concentrados numa única base de código. Esta concentração traduz-se, igualmente, numa curva de aprendizagem mais reduzida para as equipas de desenvolvimento, o que representa uma vantagem significativa em contextos de *onboarding* rápido ou em projetos com orçamentos limitados. Este tipo de arquitetura continua a ser uma escolha tecnicamente justificável quando se privilegia a previsibilidade operacional e a eficiência de recursos computacionais.

Por outro lado, os autores em [18] referem que a arquitetura monolítica apresenta limitações estruturais que não podem ser negligenciadas, sobretudo em projetos de longo prazo. Entre as principais dificuldades destaca-se a limitação da escalabilidade seletiva, dado que não é possível escalar apenas os componentes que registam maior carga, toda a aplicação precisa de ser replicada, o que implica custos operacionais adicionais e desperdício de recursos. A manutenção torna-se também mais desafiante, uma vez que modificações pontuais podem afetar módulos não relacionados, aumentando o risco de introdução de regressões. Por outro lado, estas dificuldades contribuem para ciclos de desenvolvimento mais longos e maior complexidade na realização de testes automatizados, especialmente em ambientes com elevado grau de interdependência funcional.

De acordo com a análise apresentada em [16] acresce a estas limitações a dificuldade em adotar tecnologias heterogéneas num mesmo sistema monolítico. A uniformidade da base de código e do ambiente de execução condiciona a introdução de novas *framework s* ou linguagens, o que representa uma barreira à inovação tecnológica. Além disso, o processo de implantação é rigidamente sequencial, o que compromete a agilidade organizacional e a capacidade de resposta a alterações frequentes nos requisitos. Estes fatores tornam a arquitetura monolítica particularmente vulnerável em contextos de elevada volatilidade ou em ambientes onde a escalabilidade horizontal é uma exigência estruturante.

No entanto, como estratégia de mitigação dos constrangimentos da arquitetura monolítica tradicional, tem vindo a emergir o conceito de monólito modular. Esta abordagem defende a organização da aplicação em componentes internos bem definidos, com separação clara de responsabilidades e encapsulamento lógico, mantendo-se, ainda assim, uma única unidade de execução e implantação. A modularização pode ainda contribuir para a redução da dívida técnica acumulada ao longo do tempo,

favorecendo a sustentabilidade evolutiva do sistema. Neste sentido, modularizar um monólito constitui não apenas uma prática de engenharia recomendada, mas também uma estratégia de futuro, sobretudo para organizações que ponderam escalar os seus sistemas gradualmente. A literatura recente tem sublinhado que o sucesso de uma arquitetura monolítica depende, em larga medida, da aplicação de boas práticas de engenharia, incluindo a utilização de testes unitários e de integração contínua, a aplicação rigorosa de princípios de design e o uso de ferramentas de análise de dependências para garantir a coerência entre módulos. Neste contexto um monólito bem desenhado pode ter um desempenho superior a sistemas distribuídos mal estruturados, sobretudo quando a latência e a consistência transacional são fatores críticos [35].

Assim, a escolha da arquitetura monolítica não deve ser encarada como tecnicamente obsoleta, mas sim como uma decisão fundamentada, cuja adequação dependerá do contexto, dos requisitos funcionais e não funcionais, da maturidade da equipa técnica e dos objetivos de negócio. Em muitos casos, a simplicidade inerente ao monólito, aliada a uma estrutura modular bem definida, pode oferecer um compromisso eficaz entre eficiência, manutenção e tempo de colocação no mercado.

2.3.2. Arquitetura Orientada a Serviços (SOA)

A Arquitetura Orientada a Serviços, tal como apresentada em [19], tem-se consolidado nas últimas décadas como um paradigma importante no desenvolvimento de software distribuído, centrado na composição de serviços independentes, reutilizáveis e interoperáveis. Este paradigma promove a redução de dependência entre as diferentes funcionalidades de negócio, permitindo uma maior flexibilidade na integração de sistemas heterogéneos e facilitando a manutenção evolutiva das aplicações empresariais [20]. Os serviços são tipicamente acedidos através de interfaces padronizadas, geralmente descritas por linguagens como WSDL (*Web Services Description Language*), e utilizam protocolos difundidos como SOAP (*Simple Object Access Protocol*) e REST (*Representational State Transfer*), o que permite interoperabilidade entre diferentes plataformas tecnológicas[21]. Neste contexto, segundo [22] um serviço representa uma unidade de trabalho realizada por um fornecedor de serviços para atender às necessidades específicas de um consumidor de serviços. Ambos os papéis, fornecedor e consumidor, são desempenhados por agentes de software que operam em nome dos seus respetivos proprietários.

Nos últimos anos, observou-se uma expansão significativa da aplicação da SOA em contextos emergentes como a computação em nuvem, Internet das Coisas (IoT), e sistemas ciberfísicos, refletindo a capacidade adaptativa deste paradigma perante desafios tecnológicos contemporâneos

[23]. Especificamente, a combinação da SOA com a computação em nuvem revelou-se estratégica, permitindo que serviços fossem disponibilizados como recursos sob demanda, otimizando o provisionamento e assegurando uma escalabilidade eficiente em aplicações de grande porte e alta demanda [24]. Um estudo recente, apresentado em [25] confirma que a SOA tem contribuído significativamente para modernizar aplicações empresariais baseadas em nuvem, especialmente aquelas relacionadas com a gestão da cadeia de abastecimento e ERP (*Enterprise Resource Planning*), promovendo uma maior agilidade operacional e adaptabilidade às mudanças do mercado.

A utilização da SOA no contexto da IoT também tem sido bastante documentada, particularmente devido à necessidade de gerir grandes volumes de dados heterogéneos provenientes de dispositivos conectados, como sensores e atuadores. O encapsulamento desses dispositivos como serviços independentes facilita a integração de dados e a interoperabilidade, simplificando consideravelmente a complexidade inerente aos sistemas IoT [26]. Em [27] é referenciado que a aplicação do paradigma SOA em ambientes IoT melhora a eficiência operacional, permitindo arquiteturas distribuídas robustas e adaptativas, especialmente relevantes em contextos industriais e urbanos.

Nos sistemas ciberfísicos, particularmente associados à Indústria 4.0, a aplicação da SOA tornou-se importante para alcançar a necessária interoperabilidade e flexibilidade produtiva. Pesquisas indicam que a integração de dispositivos físicos e componentes digitais por meio da SOA promove uma maior facilidade na reconfiguração dinâmica dos processos industriais, garantindo uma resposta mais ágil e eficiente às alterações nas condições operacionais ou demandas do mercado [28] . Além disso, como referido no trabalho apresentado em [29], a SOA tem sido particularmente eficaz ao permitir a integração suave de sistemas legados com tecnologias emergentes, preservando investimentos anteriores e reduzindo riscos operacionais.

Contudo, apesar das múltiplas vantagens oferecidas pela SOA, a sua implementação prática enfrenta diversos desafios críticos. Entre eles, a gestão dos serviços destaca-se como uma questão fundamental, exigindo um controlo rigoroso do ciclo de vida dos serviços, gestão de versões e estratégias claras para monitorização do desempenho e segurança [19]. Conforme alertado pelos autores no trabalho apresentado [30] o risco da proliferação descontrolada de serviços pode levar ao fenómeno conhecido como "*spaghetti services*", onde a ausência de uma gestão eficaz gera redundância, complexidade excessiva e degradação gradual da arquitetura do sistema.

A segurança dos serviços é outro aspeto de elevada relevância, especialmente porque as interfaces públicas aumentam a exposição dos sistemas a ataques e ameaças externas. Dessa forma, a implementação de robustos mecanismos de segurança como *WS-Security* e criptografia torna-se imperativa para assegurar a integridade e confidencialidade dos dados trocados entre serviços distribuídos [31]. Pesquisas enfatizam a importância de se adotarem modelos de autenticação e autorização baseados em *tokens* ou certificados digitais como medida preventiva essencial contra ataques frequentes, tais como a negação de serviço (DoS) e interceção de dados [32].
Finalmente, a integração de serviços em arquiteturas híbridas, onde coexistem infraestruturas locais e serviços em nuvem, constitui outro desafio considerável. Estudos, tais como o apresentado em [33], apontam que questões relacionadas à latência, compatibilidade tecnológica e gestão eficaz dos dados distribuídos exigem estratégias bem estruturadas e controlo operacional para mitigar possíveis impactos negativos sobre a performance global dos sistemas.

Em conclusão, a Arquitetura Orientada a Serviços permanece um paradigma essencial e atual, destacando-se pela sua capacidade em oferecer soluções flexíveis, interoperáveis e escaláveis para sistemas complexos distribuídos. Embora os desafios na gestão estratégica, segurança e integração sejam consideráveis, abordagens modernas têm permitido uma implementação bem sucedida da SOA em diversos contextos organizacionais e tecnológicos. À medida que surgem novas tendências, como a inteligência artificial integrada aos serviços, espera-se que a relevância e a aplicação prática da SOA continuem a expandir-se significativamente no futuro próximo [19] .

2.3.3. Arquitetura de Microserviços

A arquitetura de microserviços tem-se consolidado, ao longo da última década, como uma resposta aos desafios impostos pela crescente complexidade, escalabilidade e necessidade de agilidade nos sistemas de software modernos. Este modelo estrutural privilegia a decomposição de sistemas em unidades autónomas e com baixa dependência entre si, cada uma responsável por um conjunto de funcionalidades do domínio de negócio. Estes serviços comunicam entre si através de protocolos leves, como *HyperText Transfer Protocol* (HTTP) ou mensagens assíncronas, e são implantados de forma independente [34].

A definição formal de arquitetura de software, tal como proposta por [35], refere-se à “estrutura ou estruturas de um sistema, compreendendo componentes de software, as propriedades visíveis externamente desses componentes e os relacionamentos entre eles”. Esta definição aplica-se diretamente à arquitetura de microserviços, uma vez que esta promove a visibilidade e autonomia dos

componentes, fomenta a separação de responsabilidades e apoia a evolução contínua do sistema. A utilização de mecanismos de arquitetura como redundância para tolerância a falhas, particionamento para desempenho e isolamento para segurança é essencial na composição de microserviços bem desenhados.

Conforme Newman [34] explica, os microserviços são serviços pequenos, autossuficientes, organizados em torno de capacidades de negócio, que comunicam entre si usando mecanismos leves. Richardson [36] reforça a necessidade de que cada microserviço seja alinhado com os limites de contexto definidos por práticas de Domain-Driven Design, de forma a preservar a coesão funcional. Esta abordagem é suportada por uma forte base conceptual na orientação a serviços, mas diferencia-se por uma granularidade mais fina, automatização extensiva e independência de implantação [37].

A Figura 2 apresenta uma arquitetura simplificada baseada em microserviços. O cliente interage com o sistema por meio de um *gateway*, que direciona as requisições para os microserviços apropriados. Cada microserviço é independente e pode utilizar diferentes bases de dados. A comunicação assíncrona entre serviços é realizada por meio de um *message broker* e utiliza o padrão *publish/subscribe*, o que permite o desacoplamento e aumenta a escalabilidade e a resiliência do sistema.

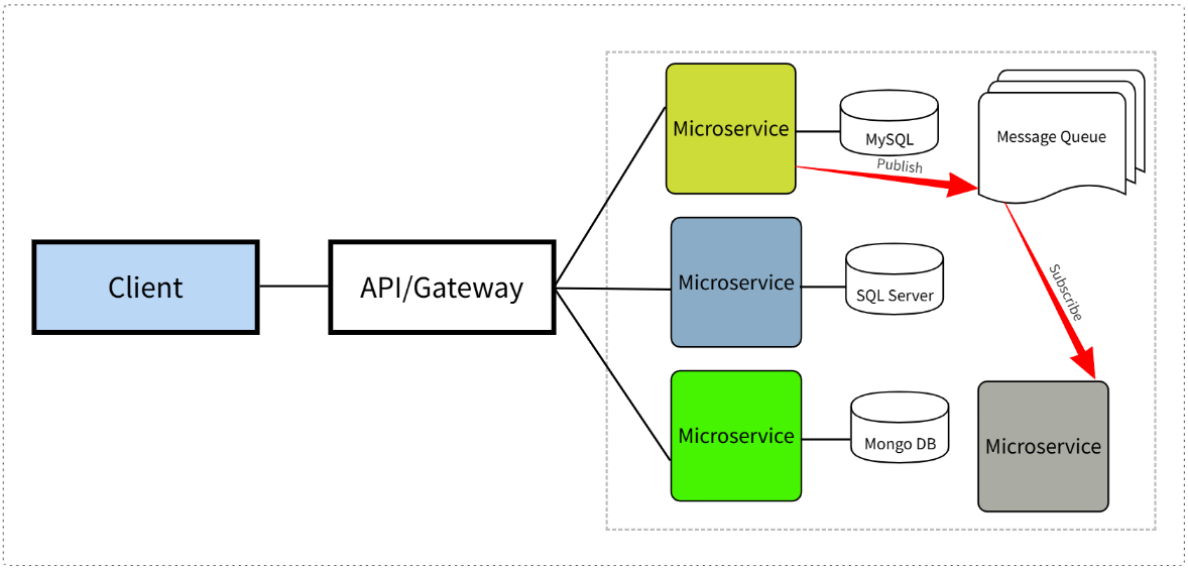


Figura 2 - Exemplo de arquitetura de microserviços

Do ponto de vista organizacional, a arquitetura de microserviços permite uma maior autonomia das equipas, que podem trabalhar em paralelo em diferentes serviços sem dependências rígidas. Este alinhamento entre arquitetura técnica e estrutura organizacional foi destacado por [38], que sublinha a correlação entre a maturidade da cultura *DevOps* e o sucesso na adoção de microserviços em

ambientes empresariais. De forma semelhante, os autores em [39] identificam que os padrões de arquitetura e práticas de migração são determinantes na transição de arquiteturas monolíticas para ambientes baseados em microserviços.

Entre os benefícios técnicos mais frequentemente referidos encontram-se a escalabilidade horizontal que permite escalar apenas os serviços que enfrentam maior carga, a resiliência operacional, possibilitada pelo isolamento de falhas, e a flexibilidade tecnológica, uma vez que diferentes serviços podem utilizar linguagens, *framework s* e bases de dados distintas [34]. Adicionalmente, a capacidade de entrega contínua é fortalecida pela independência dos ciclos de vida dos serviços, permitindo atualizações frequentes e de baixo risco, suportadas por práticas de *Continuous Integration/Continuous Deployment* (CI/CD).

No entanto, a adoção da arquitetura de microserviços também introduz desafios significativos. A gestão de um número elevado de serviços impõe complexidade adicional ao nível da orquestração, monitorização, controlo de versões de *Application Programming Interface* (API) e segurança[40], [39]. A coordenação de transações entre serviços requer padrões como sagas (padrão de transações distribuídas) e mecanismos de compensação, uma vez que a consistência forte nem sempre é possível num sistema distribuído. Kleppmann [40], ao abordar os sistemas intensivos em dados, enfatiza a necessidade de compreender os *trade-offs* entre disponibilidade, latência e consistência, centrais em ambientes baseados em microserviços.

A monitorização e observabilidade emergem como componentes essenciais para garantir a fiabilidade operacional. Ferramentas como Prometheus, *Grafana*, Jaeger e OpenTelemetry têm sido muito utilizadas para recolha de métricas, *logging* estruturado e *tracing* distribuído [41]. Willard e Hutson destacam ainda a utilização de IA para automatizar decisões de escalabilidade e alocação dinâmica de recursos, introduzindo o conceito de microserviços inteligentes, que ajustam o seu comportamento em função do contexto operacional. A integração entre microserviços e técnicas de IA representa um vetor de inovação. Esta sinergia não só contribui para a automatização operacional como também potencia a adaptabilidade dos sistemas às mudanças do ambiente. Exemplos incluem a deteção preditiva de padrões de uso, reconfiguração automática de topologias de serviços e a personalização de fluxos de negócio em tempo real [42].

Embora a arquitetura de microserviços represente uma evolução natural face às limitações das arquiteturas monolíticas e mesmo da SOA, a sua adoção deve ser ponderada. Em [37] é salientado que

o entusiasmo em torno dos microserviços pode levar a cenários de *over-engineering*, nos quais a complexidade supera os ganhos efetivos, especialmente em sistemas de menor escala. É essencial avaliar a maturidade técnica da organização, a criticidade do domínio de negócio e a disponibilidade de ferramentas de suporte antes de avançar com uma transição completa.

Em síntese, a arquitetura de microserviços constitui um modelo capaz de responder aos requisitos de escalabilidade, flexibilidade e resiliência dos sistemas contemporâneos. A sua implementação eficaz requer uma abordagem sistemática, suportada por princípios sólidos de engenharia de software, boas práticas operacionais e uma visão estratégica clara. Quando corretamente aplicada, esta arquitetura habilita as organizações a enfrentar com sucesso os desafios de um mercado digital em constante transformação.

2.3.4. Análise Comparativa das Arquiteturas Monolítica, Orientada a Serviços e de Microserviços

A análise comparativa de paradigmas de arquitetura é essencial para fundamentar decisões estratégicas no desenvolvimento de sistemas. Com base na caracterização das arquiteturas monolítica, SOA e de microserviços, esta secção sintetiza os principais aspetos entre os três modelos, com base em critérios como modularidade, escalabilidade, dependência entre componentes, comunicação e complexidade operacional.

A arquitetura monolítica caracteriza-se por uma estrutura única e indivisível, onde todos os componentes da aplicação, interface, lógica de negócio e acesso a dados estão encapsulados num único executável. Esta abordagem simplifica o desenvolvimento inicial, mas dificulta a manutenção e a evolução em sistemas de grande escala, sendo comum que alterações num único módulo exijam o *rebuild* e o *redeploy* de toda a aplicação[43]. Em contraste, a arquitetura orientada a serviços organiza o sistema em serviços autónomos, reutilizáveis e interoperáveis, que comunicam entre si através de protocolos padronizados. Cada serviço oferece uma funcionalidade específica e é registado num barramento central, o *Enterprise Service Bus* (ESB), que coordena a comunicação e a orquestração dos serviços, promovendo a interoperabilidade entre sistemas heterogéneos [44],[45].

A arquitetura de microserviços vai além da SOA ao decompor a aplicação em serviços ainda mais pequenos, coesos e independentes. Cada microserviço tem o seu ciclo de vida, a sua própria base de dados, e pode ser desenvolvido, testado, implantado e escalado de forma autónoma. Esta

granularidade e autonomia contribuem para maior coesão interna dos serviços e componentes mais autónomos entre si, aumentando a flexibilidade [46],[47].

A escalabilidade nos sistemas monolíticos é limitada, pois qualquer aumento de carga exige a replicação da aplicação como um todo, mesmo que apenas uma funcionalidade esteja a consumir mais recursos. Esta abordagem resulta em desperdício de recursos computacionais e torna-se ineficiente à medida que a complexidade da aplicação aumenta [43]. Na SOA, os serviços podem ser escalados individualmente, o que representa uma melhoria significativa. Contudo, a dependência do ESB e a partilha de recursos comuns podem criar gargalos e comprometer a performance do sistema como um todo [44]. Os microserviços, por seu lado, oferecem escalabilidade granular, permitindo que apenas os serviços críticos sejam replicados, o que resulta numa utilização mais eficiente da infraestrutura e proporciona maior resiliência e elasticidade do sistema [46]. Em termos de desenvolvimento e *deploy*, a arquitetura monolítica apresentam elevada rigidez. Qualquer alteração, mesmo pequena, implica a recompilação de toda a aplicação, o que torna os ciclos de entrega longos e arriscados [43].

A SOA introduz algum nível de independência, dado que os serviços podem teoricamente ser atualizados isoladamente. No entanto, na prática, a existência de dependências partilhadas, como bases de dados ou *middleware* centralizado, limita esta autonomia [45]. Já nos microserviços, cada equipa pode gerir o seu serviço de forma totalmente independente, com *pipelines* de integração e entrega contínua, o que permite maior agilidade e ciclos de desenvolvimento rápidos e seguros [47],[48].

Relativamente à comunicação entre componentes, a arquitetura monolítica utiliza chamadas internas diretas entre módulos, que são rápidas e simples, mas não permitem a separação de responsabilidades em tempo de execução[43].

Na SOA, a comunicação é geralmente realizada com base em mensagens XML, utilizando o protocolo SOAP sobre HTTP, e é frequentemente mediada pelo ESB. Este modelo oferece robustez e padronização, mas introduz latência e complexidade adicional[44].

A arquitetura de microserviços privilegia comunicações leves e flexíveis, através de REST, gRPC ou mensagens assíncronas com ferramentas como Kafka ou RabbitMQ. Além disso, o uso de API Gateways permite abstrair a complexidade da comunicação entre serviços, facilitando a gestão da segurança, da gestão de versões e da monitorização [46],[47].

A simplicidade de operação é uma das principais vantagens da arquitetura monolítica, especialmente em contextos com equipas pequenas e projetos de baixa complexidade. No entanto, à medida que o sistema cresce, torna-se cada vez mais difícil isolar responsabilidades, gerir dependências e mitigar regressões funcionais [43].

A SOA, apesar de promover modularidade, requer uma infraestrutura especializada e equipas com competências específicas, particularmente para a gestão do ESB, a orquestração de serviços e a garantia de segurança em ambientes distribuídos[45]. Os microserviços, embora flexíveis, implicam uma complexidade operacional muito superior. A gestão de dezenas ou centenas de serviços independentes exige soluções de descoberta de serviços, tolerância a falhas, *logging* distribuído e orquestração com plataformas como o *Kubernetes*. Esta abordagem é viável sobretudo em organizações com maturidade técnica elevada e processos de DevOps bem estabelecidos [47], [48].

Por fim, no que diz respeito à aplicabilidade, a arquitetura monolítica continua a ser uma escolha válida para aplicações de pequena ou média dimensão, com requisitos estáveis e equipas reduzidas [43]. A SOA mostra-se particularmente eficaz em ambientes empresariais onde é necessária a integração de sistemas legados, a reutilização de funcionalidades comuns e o cumprimento de requisitos normativos [44],[45]. Já os microserviços são especialmente adequados a contextos que exigem inovação contínua, escalabilidade elástica, entrega frequente e autonomia das equipas, como em soluções cloud-*native*, aplicações móveis e plataformas digitais complexas[46], [47].

A Tabela 1 sintetiza, de forma comparativa, os principais aspetos das arquiteturas monolítica, orientada a serviços (SOA) e de microserviços, com base na análise desenvolvida nesta secção.

CRITÉRIO	MONOLÍTICA	SOA	MICROSERVIÇOS
MODULARIDADE	Estrutura única, pouca modularidade em tempo de execução.	Serviços autónomos e reutilizáveis, mas dependentes do ESB.	Serviços pequenos, coesos e autónomos.
ESCALABILIDADE	Limitada; replicação da aplicação inteira necessária.	Serviços escaláveis individualmente, mas ESB pode ser gargalo.	Escalabilidade granular; apenas serviços críticos são replicados.
DEPENDÊNCIA ENTRE COMPONENTES	Elevada; todos os módulos dependem do mesmo executável.	Dependência no ESB e em middleware centralizado.	Baixa; serviços independentes com bases de dados próprias.
COMUNICAÇÃO	Chamadas internas diretas entre módulos.	Mensagens XML via SOAP, mediadas por ESB.	Protocolos leves (REST, gRPC), mensagens assíncronas (Kafka, RabbitMQ).
COMPLEXIDADE OPERACIONAL	Baixa; fácil operação em projetos pequenos.	Moderada; requer infraestrutura especializada e gestão do ESB.	Alta; exige orquestração, descoberta de serviços e monitorização distribuída.
DESENVOLVIMENTO E IMPLEMENTAÇÃO	Rígido; alterações implicam rebuild e redeploy total.	Autonomia parcial; dependências partilhadas limitam independência.	Autónomo, cada equipa gere o seu serviço e *pipeline* CI/CD.
FLEXIBILIDADE TECNOLÓGICA	Limitada; linguagem e tecnologias partilhadas.	Melhor que monolítica, mas com middleware comum.	Alta; cada serviço pode usar tecnologias e bases de dados distintas.
ADEQUAÇÃO A CONTEXTOS	Aplicações pequenas ou médias, requisitos estáveis.	Integração de sistemas legados, reutilização e requisitos normativos.	Sistemas cloud-native, inovação contínua, plataformas digitais complexas.
MANUTENÇÃO E EVOLUÇÃO	Complexa em larga escala; risco de regressões.	Facilita reutilização, mas middleware pode complicar.	Facilita evolução independente e entregas frequentes.

Tabela 1 - Comparação entre as arquiteturas monolítica, SOA e microserviços

Conclui-se, assim, que cada paradigma apresenta vantagens e limitações, devendo a sua escolha ser orientada não apenas por critérios técnicos, mas também por fatores organizacionais, culturais e estratégicos. Enquanto a simplicidade das arquiteturas monolíticas as torna acessíveis, a modularidade da SOA e a autonomia dos microserviços oferecem capacidades superiores, embora à custa de maior complexidade e exigência de maturidade tecnológica.

2.4. Casos Reais de Digitalização de Processos nas Organizações

A transformação digital de processos organizacionais tem sido implementada em diversos setores, permitindo identificar padrões de adoção e benefícios obtidos. Esta secção apresenta três casos de

diferentes indústrias que mostram abordagens específicas para digitalização e automatização de *workflows*.

2.4.1. Deutsche Telekom - Telecomunicações

A Deutsche Telekom, empresa no setor das telecomunicações na Europa, enfrentou sérias dificuldades técnicas e operacionais devido à sua dependência em tecnologias antigas, como por exemplo BPEL (*Business Process Execution Language*). A linguagem BPEL, que foi concebida para definir e executar processos de negócio estruturados, mostrou-se limitada no contexto atual, sobretudo pela sua rigidez face às exigências crescentes de agilidade e rapidez nas alterações. Estas limitações refletiam-se diretamente nos ciclos demorados de atualização, que frequentemente ultrapassavam um ano, afetando negativamente a capacidade competitiva da organização. Para enfrentar essas limitações, em 2017 a Deutsche Telekom iniciou uma transformação tecnológica, adotando uma nova arquitetura baseada em microserviços. Neste processo, optou pela utilização do Camunda como núcleo central de orquestração, integrando-o diretamente nos seus serviços construídos em Java e utilizou a *framework* Spring Boot com *Kubernetes*. Esta mudança permitiu fragmentar os antigos sistemas monolíticos em unidades menores e independentes, que poderiam ser atualizadas de forma isolada, com menor risco operacional e maior rapidez. A arquitetura escolhida tornou possível a adaptação incremental e flexível às necessidades do negócio, garantindo ainda uma maior resiliência dos processos críticos da empresa [49].

Durante a migração tecnológica, um dos grandes desafios surgiu relacionado à automatização de processos via RPA (Robotic Process Automation). Anteriormente, a Deutsche Telekom utilizava cerca de três mil *bots* distribuídos por diferentes plataformas, o que provocava uma grande complexidade técnica e operacional, além de custos de manutenção elevados devido à fragilidade desses *bots* frente a mudanças constantes na interface gráfica das aplicações. Para superar esta dificuldade, a empresa criou uma plataforma interna, denominada "OREO", integrada com o motor Camunda. Essa plataforma passou a coordenar as diferentes automatizações, sejam estas executadas por bots, API ou por intervenções humanas e usou diagramas BPMN e *Decision Model and Notation* (DMN) para garantir clareza e padronização dos processos. Além disso, utilizou a funcionalidade *external tasks* para desacoplar a execução dos serviços da lógica central de orquestração, o que promoveu maior estabilidade e facilidade de gestão das operações distribuídas [49].

A transição tecnológica gerou resultados significativos e rapidamente observáveis. Até ao final de 2019, foram automatizados aproximadamente 450 processos, o que permitiu gerir mais de 40 milhões

de transações por ano. Estimativas realizadas apontaram para uma redução significativa nos custos operacionais anuais que ultrapassam 93 milhões de euros, com um retorno sobre o investimento estimado de aproximadamente três euros por cada euro gasto no projeto. Além dos benefícios financeiros, registaram-se melhorias na eficiência operacional, na capacidade de auditoria dos processos. Contudo, é importante realçar que a transformação não se limitou às melhorias técnicas. O sucesso do projeto dependeu de uma significativa mudança organizacional e cultural. A Deutsche Telekom adotou metodologias ágeis estruturadas, nomeadamente o Scaled Agile *Framework* (SAFe), complementadas por práticas modernas de engenharia como a integração e entrega contínua (CI/CD). Essa abordagem favoreceu uma migração gradual e controlada, minimizando riscos técnicos e facilitando a adesão das equipas envolvidas. A plataforma interna OREO também desempenhou um papel importante ao definir explicitamente critérios para a utilização estratégica de bots, API ou intervenções humanas, assegurando uma automatização coerente e sustentável ao longo do tempo [49].

Em resumo, o caso da Deutsche Telekom constitui um exemplo concreto de como uma abordagem estruturada e integrada, envolvendo tecnologias modernas, metodologias ágeis e práticas avançadas de desenvolvimento, pode transformar positivamente organizações com infraestruturas complexas e rígidas, tornando-as mais competitivas, ágeis e preparadas para responder à rápida evolução tecnológica dos mercados.

2.4.2. Banco de Occidente - Setor Financeiro

O Banco de Occidente, uma das principais instituições bancárias da Colômbia, enfrentava limitações no desempenho dos seus serviços devido à coexistência de canais presenciais e digitais. Estes processos, marcados por elevada intervenção manual, comprometiam a agilidade operacional, dificultavam a rastreabilidade e criavam barreiras ao cumprimento das exigências regulamentares e ao aumento da satisfação dos clientes. Face a este cenário, a instituição deu início a um programa de transformação digital orientada à otimização de processos, com o objetivo de aumentar a eficiência e flexibilidade das operações. Para alcançar esses objetivos, o banco implementou a plataforma Bizagi, uma solução low-code compatível com a notação BPMN 2.0, que permitiu às equipas de negócio colaborar diretamente no desenho e evolução dos fluxos automatizados. A seleção desta tecnologia baseou-se na sua capacidade de orquestrar diferentes componentes de processo, *user tasks*, regras de decisão e chamadas a sistemas externos, através de uma interface visual acessível e modular.

A arquitetura adotada permitiu integrar o Bizagi com sistemas legados e aplicações externas através de API. Esta integração viabilizou a execução coordenada dos processos num ambiente distribuído, garantindo simultaneamente a continuidade operacional e a escalabilidade do sistema. O motor de processos passou a coordenar tarefas como validações de dados, notificações automáticas e decisões condicionais, contribuindo para fluxos mais transparentes e auditáveis [50].

Com base no estudo de caso do Banco de Occidente [50], é possível identificar os seguintes elementos que sustentaram a transformação digital promovida pela instituição:

- **Modelação orientada a processos**: A utilização da notação BPMN 2.0 permitiu ao banco estruturar os seus fluxos de trabalho de forma visual, o que assegurou maior clareza, padronização e facilidade na sua manutenção e adaptação.
- **Abordagem low-code**: A plataforma Bizagi disponibilizou uma interface acessível, que permitiu a participação direta das equipas de negócio na definição e evolução dos processos, reduzindo a dependência exclusiva das equipas de desenvolvimento.
- **Integração com sistemas legados e externos**: A solução adotada possibilitou a comunicação entre o motor de processos Bizagi e os sistemas *core* do banco, bem como com serviços externos, através de conectores configuráveis, assegurando continuidade operacional e interoperabilidade.
- **Automatização de decisões**: Embora não detalhado com a terminologia DMN, o estudo refere a possibilidade de modelar regras de negócio diretamente nos fluxos, permitindo decisões automáticas e consistentes com base em condições pré-definidas.
- **Monitorização e visibilidade operacional**: A solução implementada incluiu painéis de controlo em tempo real (*dashboards*), que forneceram visibilidade clara sobre o estado de execução dos processos e indicadores de desempenho, facilitando a gestão e a tomada de decisão.
- **Supervisão e controlo operacional**: O sistema passou a permitir um controlo centralizado sobre os fluxos de trabalho, contribuindo para maior transparência, rastreabilidade e conformidade com os requisitos operacionais e regulatórios.

A aplicação Bizagi trouxe resultados em termos de desempenho e eficiência. Os processos passaram a ser executados em menos tempo, com menor necessidade de intervenção humana e maior capacidade de adaptação a requisitos operacionais. A implementação contribuiu também para um aumento da autonomia das equipas de negócio na gestão e otimização dos fluxos, promovendo uma cultura de melhoria contínua e reduzindo a dependência exclusiva da equipa de desenvolvimento.

2.4.3. Empresa Industrial (Estudo Anónimo) – Indústria Transformadora

A investigação realizada e apresentada em [51] incide sobre a implementação de uma solução de gestão de processos de negócio numa organização industrial de média dimensão, localizada na Bósnia e Herzegovina. Esta empresa enfrentava entraves significativos ao nível da eficiência do planeamento de produção, motivados, sobretudo, pela utilização intensiva de documentos em suporte físico, pela execução de tarefas manuais fragmentadas entre departamentos e pela ausência de rastreabilidade nos registos operacionais. Estas fragilidades refletiam-se numa elevada incidência de erros, baixa produtividade e dificuldade em garantir a conformidade dos procedimentos internos.

Com o intuito de mitigar essas limitações, foi delineada uma estratégia de digitalização orientada à reformulação do ciclo de planeamento. Para tal, foi adotada a plataforma Bonita, cuja implementação se estruturou da seguinte forma:

- Levantamento e análise dos fluxos existentes;
- Modelação do processo alvo;
- Desenvolvimento da solução com recurso a tecnologia BPM;
- Monitorização e melhoria contínua com base em métricas de desempenho.

A transformação digital incidiu sobre a totalidade do circuito documental do planeamento, abrangendo desde o pedido inicial do cliente até à entrega final do produto. Foram sistematizados dez documentos essenciais, entre os quais se incluem ordens de fabrico, listas de materiais, folhas de operação, relatórios de qualidade e registos de armazém, integrados num processo único, automatizado e acessível digitalmente. A modelação do fluxo foi realizada com recurso à notação BPMN 2.0, operacionalizada no Bonita Studio, enquanto a execução foi disponibilizada às várias equipas através do Bonita Web Portal.

A solução implementada permitiu não apenas digitalizar os procedimentos, mas também reestruturar o modelo de gestão dos processos. A organização do fluxo em *swimlanes* funcionais assegurou a distribuição clara de responsabilidades entre as seis áreas envolvidas (clientes, vendas, produção, operações, qualidade e armazém), promovendo a transparência e a colaboração. As interfaces desenvolvidas possibilitaram a introdução de dados em tempo real, validados automaticamente através de contratos de processo. Adicionalmente, o sistema registou todas as interações com o processo, garantindo uma rastreabilidade completa e um suporte, para auditorias internas.

Embora o estudo não explore diretamente a adoção de microserviços, é relevante destacar que a própria plataforma Bonita BPM possui uma arquitetura compatível com esse paradigma, suportando chamadas REST, conectores externos reutilizáveis, execução assíncrona de tarefas e lógica distribuída, o que a torna adequada para cenários mais exigentes de integração tecnológica[52].

Em síntese, o desenho técnico da solução implementada em [51] inclui:

- Modelação do processo com BPMN 2.0;
- *Swimlanes* funcionais para organização das seis unidades operacionais envolvidas;
- Utilização do Bonita Studio como ambiente de desenvolvimento do fluxo;
- Modelação de dados empresariais (BDM) para suporte das entidades documentais;
- Interfaces web interativas, adaptadas ao perfil de cada interveniente;
- Contratos de processo com validação de dados à entrada de cada tarefa;
- Gestão de acessos diferenciada por função e responsabilidade;
- Execução distribuída via portal web, acessível a todos os departamentos;
- Registo automático de ações, permitindo controlo e rastreabilidade.

Com a entrada em funcionamento do novo sistema, verificou-se uma redução significativa no tempo necessário para processar a documentação relativa ao planeamento, acompanhada de uma descida acentuada do número de erros causados por falhas humanas. A centralização da informação permitiu reforçar a visibilidade sobre o estado dos processos e facilitou a deteção e resolução de ineficiências. O projeto proporcionou ainda uma base para futuras integrações com sistemas ERP, sustentando a evolução digital da empresa de forma escalável e controlada.

Este caso evidencia o potencial transformador das plataformas BPM na modernização de organizações industriais, mesmo em contextos com recursos limitados. A opção pela solução Bonita BPM demonstrou ser uma arquitetura tecnológica adaptável, com capacidade de crescimento e integração em ecossistemas maiores. Ainda que tenham sido identificados alguns desafios técnicos, como o ajuste inicial dos requisitos operacionais ao modelo BPMN, o esforço de configuração das interfaces web e a limitação nas integrações externas na fase inicial, estes foram superados através da colaboração contínua com os utilizadores [51].

2.5. Análise e Comparação de Plataformas de Automação de Processos de Negócio

Esta secção tem como objetivo analisar e comparar três plataformas bastante utilizadas na automação de processos de negócio, nomeadamente Camunda**,** Bizagi e Bonita BPM. A escolha recaiu sobre estas ferramentas por refletirem abordagens distintas no que diz respeito à arquitetura, ao modelo de licenciamento e ao grau de abertura tecnológica. Além disso, são soluções com boa documentação, com presença significativa em projetos reais e comunidades técnicas, o que permite uma análise fundamentada e comparável.

A metodologia adotada inclui uma análise individual de cada plataforma, centrada na sua estrutura técnica e nas funcionalidades descritas na documentação oficial, complementada por experiências partilhadas por utilizadores. Por fim, é apresentada uma comparação com base em critérios como arquitetura, usabilidade, integração, escalabilidade e licenciamento.

2.5.1. Camunda

Nos últimos anos, o Camunda tem-se evidenciado como uma plataforma para automação de fluxos de trabalho e de tomada de decisões, oferece suporte aos padrões BPMN e DMN utilizados respetivamente para a modelação de processos de negócio e para a definição de regras de decisão [53].

Historicamente, o Camunda 7 apresentava um motor leve, desenvolvido em Java, acessível por meio de API REST ou Java com integração com *framework s* muito utilizados, como Spring e Java EE. A evolução para o Camunda 8 marcou um avanço significativo com a introdução do motor *Zeebe*, baseado numa arquitetura distribuída e comunicação via gRPC (*Google Remote Procedure Calls*). Esta transição permitiu ganhos significativos de desempenho, escalabilidade e resiliência, particularmente em contextos *cloud-native* e com recurso em ambientes *Kubernetes* [53].

Segundo a documentação oficial da Camunda, os principais componentes da plataforma Camunda 8 têm funções distintas e complementares na automatização de processos de negócio [53]:

- ***Zeebe***: Motor de *workflow* responsável pela execução de modelos BPMN. Foi concebido para operar em ambientes distribuídos, permitindo a orquestração de processos de forma escalável, resiliente e assíncrona.

- ***Tasklist***: Aplicação web destinada aos utilizadores finais que participam em *user tasks* dentro dos processos. Permite funções como visualizar e concluir tarefas atribuídas, garantindo a integração entre o sistema e a atuação humana.
- ***Operate***: Ferramenta de monitorização operacional que fornece visibilidade sobre a execução dos processos. Permite consultar o histórico, identificar incidentes e intervir manualmente quando necessário para garantir a continuidade da execução.
- ***Modeler***: Editor gráfico (disponível em versão desktop e web) que permite modelar processos em BPMN e regras de decisão em DMN. Inclui funcionalidades de validação e simulação de fluxo (*token simulation*), facilitando a verificação lógica dos modelos.
- ***Identity***: componente de gestão de identidade e autenticação, utilizado para configurar permissões de acesso e autenticar utilizadores. Suporta integração com sistemas externos como LDAP e OpenID Connect.
- ***Optimize***: Plataforma de análise de dados e relatórios que permite acompanhar métricas de desempenho dos processos automatizados. Suporta a criação de *dashboards* personalizados para tomada de decisões baseada em dados.
- ***Forms***: Ferramenta integrada de criação de formulários utilizada para interação com os utilizadores nos pontos de decisão ou tarefa dos processos. Permite criar formulários de forma visual, sem necessidade de código, para captura estruturada de dados.
- ***Connectors***: Componentes reutilizáveis que permitem integrar serviços e sistemas externos diretamente em modelos BPMN, simplificando a conectividade com API e plataformas de terceiros.

O Camunda 8 encontra-se disponível numa versão gratuita, denominada Community Edition *(Self-Managed Free Edition*), indicada principalmente para atividades de desenvolvimento e testes. Esta versão oferece acesso aos principais componentes da plataforma, nomeadamente *Zeebe*, *Tasklist*, *Operate*, *Modeler* e Identity. Desde a versão 8.6 (outubro de 2024), o uso da plataforma em ambientes de produção requer a subscrição da *Enterprise Edition*, que disponibiliza suporte técnico, capacidades de *clustering*, atualizações sem *downtime* e monitorização avançada [54].

O Camunda 8 adota uma arquitetura orientada a microserviços e suporta a orquestração de serviços desacoplados através do motor *Zeebe*. Esta abordagem facilita a escalabilidade horizontal, a tolerância a falhas e a gestão do estado distribuído dos processos. Os componentes comunicam de forma assíncrona e podem ser implementados de forma independente, com suporte para integração por gRPC, REST e brokers de mensagens como Kafka. A plataforma também suporta cenários de integração

híbrida, permitindo que empresas combinem serviços legados com microserviços. Esta flexibilidade é implementada através de padrões como *external task workers* e conectores reutilizáveis, tornando o Camunda adequado a ambientes heterogéneos e iniciativas de modernização gradual [53].

A documentação oficial da Camunda destaca diversas vantagens da plataforma, entre as quais se inclui a flexibilidade proporcionada pelos padrões BPMN e DMN, que facilitam a personalização e a adaptação contínua dos processos às necessidades das organizações. A arquitetura *headless* permite uma integração fluida com diferentes sistemas e o desenvolvimento de soluções ajustadas a contextos específicos. O acesso ao código-fonte e o seu modelo extensível reforçam ainda mais a capacidade de orquestrar microserviços de forma controlada [53].

Além das vantagens descritas na documentação oficial, a plataforma PeerSpot [60] reúne *feedback* de utilizadores que destacam, entre outros aspetos, a facilidade na modelação de processos por perfis não técnicos, a integração com diversas linguagens e tecnologias sem necessidade de alterações significativas facilidade de escalar horizontalmente com containers, redução de custos operacionais e de infraestrutura, aumento da produtividade e reaproveitamento de código. Também são apontadas limitações, como a ausência de suporte a múltiplos idiomas nas interfaces *Cockpit*, Admin e *Tasklist*, a escassez de materiais formativos e de documentação aprofundada, não é um ambiente *no-code/low-code* completo, dificuldades na configuração inicial e processos de migração, bem como um modelo de licenciamento menos acessível para pequenas empresas. Estas observações refletem experiências práticas partilhadas por utilizadores da comunidade e devem ser consideradas na avaliação da adoção da ferramenta.

Assim, o Camunda 8 posiciona-se como uma solução madura e versátil para a orquestração de processos de negócio, com forte capacidade de integração e alinhamento com arquiteturas recentes. Não obstante os seus benefícios, a adoção da plataforma implica uma análise, fundamentada nas características estruturais da organização, no grau de maturidade tecnológica e nas demandas específicas do setor de atuação.

2.5.2. Bizagi

O Bizagi é uma plataforma de automação e modelação de processos de negócio baseada no padrão BPMN 2.0, concebida para facilitar a colaboração entre equipas técnicas e de negócio num ambiente *low-code*. A sua arquitetura baseia-se em três componentes principais:

- **Bizagi *Modeler***: Utilizado para a modelação gráfica de processos com funcionalidades de exportação em múltiplos formatos (Word, *Portable Document Format* (PDF), *HyperText Markup Language* (HTML), Wiki), importação de modelos BPMN, XPDL ou Visio, e simulação colaborativa;
- **Bizagi Studio**: Responsável pela definição de modelos de dados, regras de negócio, interfaces de utilizador e integrações externas;
- **Bizagi Automation Server**: Executa os processos, disponibiliza o portal de trabalho para interação com os utilizadores finais e oferece suporte a múltiplos idiomas [55].

Segundo a documentação da Bizagi [55], particularmente na secção *"Invoking REST services"*, a plataforma fornece uma API RESTful autenticada via *OAuth* 2.0, que permite operações como criação e gestão de casos, execução de tarefas, consulta de dados e ativação de subprocessos. Adicionalmente, oferece suporte a chamadas SOAP e OData (Open Data Protocol), o que aumenta a compatibilidade com diferentes ambientes tecnológicos. Na secção *"Bizagi Connectors"*, é descrito que a integração com sistemas empresariais é suportada por conectores pré-configurados para soluções como SAP (*Systems, Applications and Products in Data Processing*), Oracle, UiPath, SharePoint e ferramentas de inteligência artificial, permitindo interoperabilidade com diversas arquiteturas e com serviços expostos via HTTP. A plataforma disponibiliza ainda funcionalidades de monitorização, auditoria e controlo operacional, assegurando maior transparência e eficiência na gestão dos processos.

A infraestrutura na *cloud* é modular, *multi-tenant* e gerida diretamente pela Bizagi, conforme descrito na secção "*Building processes for the cloud* " da documentação da Bizagi [55], inclui autenticação por SAML (*Security Assertion Markup Language*) e *OAuth*, cache distribuída com *Redis*, replicação de dados operacionais para bases secundárias e isolamento entre ambientes. Esta abordagem reduz o esforço técnico das organizações utilizadoras e facilita a manutenção e atualização da plataforma.

Contudo, segundo a documentação da Bizagi em [55], na secção "*Control list",* regista várias limitações: os uploads de ficheiros estão limitados a 25 MB ou a 230 segundos; chamadas externas síncronas estão sujeitas ao mesmo tempo máximo; conectores impõem um tamanho limite de 50 MB por requisição; e é exigida compatibilidade total entre as versões do Bizagi Studio e do *Automation* Server, impedindo atualizações parciais. A personalização do portal de trabalho está restrita a widgets e temas aprovados, sendo desaconselhada a modificação direta do HTML, *Cascading Style Sheets* (CSS) ou JavaScript.

Adicionalmente, utilizadores da plataforma PeerSpot [56] identificam diversos desafios práticos na utilização do Bizagi em contextos empresariais reais. Entre as limitações mais frequentemente reportadas destacam-se dificuldades na integração com sistemas ERP complexos, exigindo frequentemente o desenvolvimento manual de conectores personalizados em ambientes como .NET e Visual Studio, devido à ausência de suporte nativo a serviços REST com autenticação por client ID e secret. Verifica-se também degradação de desempenho em processos de grande dimensão, bem como limitações na escalabilidade horizontal, especialmente em ambientes *on-premises*, onde a distribuição de carga depende de configurações manuais com balanceadores externos. A personalização avançada da interface de utilizador é outro ponto crítico, sendo apontadas restrições na modificação do layout, nos controlos visuais e na geração de relatórios gráficos, considerados pouco flexíveis e com capacidades limitadas de visualização. Em implementações de maior escala, a estabilidade operacional da plataforma é motivo de atenção, com relatos de bugs em versões *on-premises*, dificuldades na gestão de versões entre componentes (*Studio e Automation Server*) e dependência de suporte comunitário, cuja eficácia é variável. Acrescem ainda outras limitações identificadas, como a inexistência de suporte para sistemas Linux, a ausência de funcionalidades nativas de inteligência artificial e *machine learning*, a fraca granularidade dos dados exportados nas simulações, e a falta de funcionalidades completas de gestão de casos (*case management*).

Conforme descrito na secção *'Service Level Agreement'* da documentação oficial da Bizagi [55], o modelo de licenciamento do Bizagi segue uma abordagem freemium. O Bizagi *Modeler* e Studio são disponibilizados gratuitamente para desenvolvimento local, enquanto a execução em ambiente de produção requer subscrição da edição *Enterprise*, baseada em consumo. A plataforma oferece níveis de serviço (SLA) que variam entre 99,95 % e 99,99 %, consoante o plano contratado.

O Bizagi distingue-se pela usabilidade, rapidez de adoção, conectores prontos a usar e suporte a vários idiomas, mas as limitações documentadas e os testemunhos de utilizadores reforçam a importância de uma avaliação ponderada em contextos empresariais exigentes.

2.5.3. Bonita

O Bonita BPM é uma plataforma *open source* de automatização de processos compatível com BPMN 2.0, concebida para combinar modelação visual e extensibilidade técnica. A arquitetura da plataforma Bonita assenta em três componentes principais, conforme descrito na secção *"Bonita Components"* da documentação oficial [57]:

- **Bonita Studio**: Ambiente de desenvolvimento gráfico utilizado para modelar processos de negócio, definir dados e criar aplicações baseadas em processos. Integra ferramentas de *low-code* para desenhar modelos de dados, definir conectividade com sistemas externos e desenvolver interfaces de utilizador. Inclui ainda o *UI Designer*, que permite criar formulários, páginas e layouts personalizados.
- **Bonita Runtime**: Conjunto de servidores de execução que processam as instâncias de processo em ambientes de qualificação ou produção. Pode ser instalado em máquinas físicas, virtuais, cloud ou *containers* (*Docker*), suporta configuração em *cluster*. Inclui aplicações pré-configuradas como a *Bonita User Application* (interface de tarefas para utilizadores finais), a Bonita *Administrator Application* (gestão de processos, recursos e execução), a Bonita Super *Administrator Application* (configuração técnica do Runtime), e o Application Directory (catálogo de aplicações acessíveis ao utilizador autenticado).
- **Bonita Continuous Delivery (BCD)** – Ferramenta destinada à gestão do ciclo de vida das aplicações Bonita em ambientes *Enterprise*. Facilita a automatização de *deploys*, integrações com *pipelines CI/CD* e operações DevOps através de ferramentas como *Docker*, Ansible e *Kubernetes*. Está disponível apenas nas edições por subscrição, sendo incluída por defeito em contratos adquiridos ou renovados a partir de 2021.

A comunicação com sistemas externos é assegurada com o uso de conectores reutilizáveis, descritos em "*Connectors – overview"* [57]. Estes conectores permitem aceder a API REST, SOAP, bases de dados SQL, ECM/CMIS e diretórios LDAP, com suporte a utilização assíncrona. A documentação esclarece ainda que os conectores são extensíveis via *SDK (Software Development Kit)* disponibilizado via *Maven* e podem ser associados ao início ou ao fim de tarefas ou processos para manipulação de dados ou integração com sistemas externo.

A documentação oficial distingue duas edições principais da plataforma, a *Community Edition*, gratuita e *open-source*, voltada para desenvolvimento autónomo com suporte limitado à comunidade, e a *Enterprise Edition*, com funcionalidades mais avançadas como clustering, autenticação LDAP/SSO, *dashboards* operacionais e acesso a suporte técnico profissional, conforme descrito na secção *"Bonita Editions and Licenses"* [57]. Segundo a plataforma PeerSpot [58], os utilizadores destacam a eficiência na automatização de processos, a agilidade na criação de formulários e a robustez funcional do Bonita. Contudo, são apontadas limitações significativas, nomeadamente a acentuada curva de aprendizagem, a complexidade na criação de conectores personalizados e a rigidez do UI Designer. Adicionalmente, são referidas fragilidades na documentação técnica, suporte móvel reduzido, dificuldades no

balanceamento de carga em ambientes *on premises* (instalações locais nos servidores da organização) e falta de funcionalidades nativas para *case management* (gestão de casos). No contexto da edição da comunidade, verificam-se ainda instabilidades ocasionais no ambiente de desenvolvimento, nomeadamente falhas no *workspace*, bem como a ausência de um motor de regras nativo. Embora estas limitações não comprometam a utilização da solução, reforçam a importância de uma avaliação técnica cuidada em projetos de maior complexidade.

Em síntese, o Bonita BPM representa uma boa solução, cuja adoção deve ser ponderada face aos requisitos técnicos e organizacionais de cada contexto.

2.5.4. Análise Comparativa

A análise comparativa das ferramentas Camunda, Bizagi e Bonita BPM permite identificar pontos fortes e limitações relevantes, tendo como base os seguintes critérios: Suporte a Padrões e Modelação de Processos, arquitetura, usabilidade, extensibilidade, integração, escalabilidade e modelo de licenciamento. A avaliação é sustentada na documentação oficial de cada plataforma e em relatos de utilizadores na literatura especializada.

2.5.4.1. Suporte a Padrões e Modelação de Processos

As três ferramentas suportam BPMN 2.0 como notação padrão para modelação de processos. O Camunda e o Bonita BPM também suportam DMN e UI Designer respetivamente, enquanto o Bizagi oferece funcionalidades de simulação e exportação de modelos. No entanto, apenas o Camunda oferece suporte explícito a DMN, para processos com regras de decisão complexas [53], [55], [57].

2.5.4.2. Arquitetura e desempenho

No que diz respeito à arquitetura e ao desempenho, o Camunda 8 adota uma arquitetura distribuída e orientada a microserviços, baseada no motor *Zeebe*, com suporte nativo a *Kubernetes* e comunicação por gRPC, o que permite escalabilidade, resiliência e capacidade de processamento paralelo em contextos cloud-native [53]. O Bonita Runtime também permite *deploys* em *cloud* e *containers*, mas apresenta limitações em ambientes *on-premises*, nomeadamente no balanceamento de carga e gestão de recursos[57], [58]. O Bizagi, por sua vez, embora forneça uma infraestrutura *cloud multi-tenant* gerida, enfrenta desafios de escalabilidade horizontal e degradação de desempenho em processos de grande escala, sobretudo em ambientes locais [56].

2.5.4.3. Usabilidade e abordagem low-code

No que diz respeito à usabilidade e abordagem *low-code*, o Bizagi evidencia-se pela forte orientação low-code e pela acessibilidade a perfis não técnicos, com ferramentas intuitivas como o *Modeler* e o Studio [55]. O Bonita oferece ferramentas visuais, mas com uma curva de aprendizagem acentuada e limitações no UI Designer [58]. O Camunda exige maior conhecimento técnico, não se posicionando como uma plataforma *no-code/low-code* completa[59].

2.5.4.4. Integração e extensibilidade

Relativamente à integração e extensibilidade o Camunda, Bizagi e Bonita fornecem conectores reutilizáveis e suporte a API REST/SOAP [53], [55], [57]. O Camunda oferece uma abordagem extensível via *external task workers* e conectores configuráveis, com suporte nativo a Kafka e ambientes híbridos [53]. O Bizagi dispõe de conectores para SAP, Oracle e outras plataformas, mas enfrenta limitações práticas na integração com sistemas ERP mais complexos [55], [56]. O Bonita permite extensões via SDK e integra-se com diversos sistemas externos, mas a criação de conectores personalizados é considerada complexa [57], [58].

2.5.4.5. Monitorização, gestão e inteligência analítica

No que respeita à monitorização, gestão e inteligência analítica, o Camunda disponibiliza ferramentas como o *Operate* e o *Optimize*, que permitem acompanhar a execução dos processos em tempo real e analisar o seu desempenho com *dashboards* personalizáveis [53]. O Bizagi também fornece funcionalidades de controlo e auditoria, mas apresenta limitações na flexibilidade dos relatórios gráficos disponíveis [56]. Já o Bonita BPM oferece *dashboards* operacionais apenas na edição *Enterprise*, o que restringe significativamente a capacidade de análise na versão *open-source* [57].

2.5.4.6. Modelo de licenciamento e suporte

O Camunda e o Bonita oferecem versões *open-source*, com acesso ao código-fonte, embora apresentem funcionalidades limitadas e sem acesso ao suporte técnico oficial [53], [57]. Já o Bizagi segue um modelo *freemium*, em que as ferramentas de desenvolvimento (*Modeler e Studio*) estão disponíveis gratuitamente, mas a execução em ambiente de produção exige a subscrição da edição *Enterprise* [55]. As edições *Enterprise* de todas as plataformas envolvem custos significativos e estão sujeitas a cláusulas específicas de escalabilidade, manutenção e SLA (*Service Level Agreement*). Entre as soluções analisadas, o Camunda distingue-se por dispor de uma comunidade de utilizadores mais

ativa e recursos técnicos mais atualizados, enquanto o suporte disponível nos fóruns e canais das edições *open-source* do Bonita e do Bizagi é considerado inconsistente em alguns contextos práticos [56], [58], [59].

2.5.4.7. Tabela de síntese comparativa

A tabela abaixo apresenta a validação dos critérios comparativos utilizados na análise das plataformas Camunda, Bizagi e Bonita BPM. Todos os dados foram verificados com base na documentação oficial e em fontes secundárias.

CRITÉRIO	CAMUNDA 8	BIZAGI	BONITA BPM	OBSERVAÇÕES
SUPORTE A BPMN	Sim	Sim	Sim	Impacto na modelação e interoperabilidade
SUPORTE A DMN	Sim	Não	Não	Impacto na definição de regras de negócio
ARQUITETURA	Microserviços, distribuída	Modular com execução cloud e centralizada	Modular, suporte a containers	Impacto escalabilidade e manutenção
USABILIDADE	Técnica, flexível	Intuitiva, low-code	Visual, com curva de aprendizagem acentuada	Impacto na resistência dos utilizadores
INTEGRAÇÃO	REST, gRPC, Kafka, conectores	REST, SOAP, OData, SAP	REST, SOAP, SDK	Flexibilidade de integração
ESCALABILIDADE	Elevada (cloud-native)	Limitada em ambientes locais	Parcial, com melhor desempenho em cloud	Crucial para grandes volumes de dados
MONITORIZAÇÃO / ANÁLISE	*Dashboards* via *Operate* e *Optimize*	Relatórios básicos	*Dashboards* apenas na edição *Enterprise*	Importante para a gestão
LICENCIAMENTO (MODELO)	Community + *Enterprise*	Freemium + *Enterprise*	Community + *Enterprise*	Impacto nos custos totais
ACESSO AO CÓDIGO-FONTE	Sim (Camunda 7; parcial no 8)	Não	Sim	Importante para Personalização
GRATUITO PARA PRODUÇÃO	Sim (Camunda 7); Não desde v8.6	Não	Sim (com limitações)	Impacta custos totais
SUPORTE *OPEN-SOURCE*	Ativo	Moderado	Variável	Apoio da comunidade técnica
QUALIDADE DA DOCUMENTAÇÃO	Excelente	Boa	Moderada	Acelera desenvolvimento
CASE MANAGEMENT NATIVO	Não	Não	Não	Requer desenvolvimento personalizado

Tabela 2 - Síntese comparativa das plataformas Camunda 8, Bizagi e Bonita BPM

2.5.4.8. Conclusão análise comparativa

A escolha de uma plataforma de gestão de processos de negócio deve ir além da análise funcional. É essencial considerar de forma crítica o modo como cada ferramenta se enquadra nos objetivos, restrições técnicas e capacidades da organização. Critérios como a arquitetura subjacente, a facilidade de uso, o grau de integração com sistemas existentes e o potencial de escalabilidade influenciam diretamente a eficiência operacional e a aceitação pelas equipas envolvidas.

Aspetos como o modelo de licenciamento, o acesso ao código-fonte e a qualidade da documentação técnica têm também um peso considerável, pois afetam os custos totais e a viabilidade da manutenção e evolução da solução ao longo do tempo.

À luz destes fatores, o Camunda 8 evidencia-se como uma opção particularmente sólida para contextos em que se valoriza autonomia técnica, flexibilidade da arquitetura e integração com arquiteturas cloud-native. O suporte nativo a decisões baseadas em DMN e a sua capacidade de escalar horizontalmente tornam-no adequado para organizações com requisitos complexos e ambições de transformação digital sustentada. É verdade que a adoção desta plataforma exige um maior grau de preparação técnica e que, desde a versão 8.6, o uso em produção está sujeito a licença, mas os benefícios em controlo e extensibilidade tendem a justificar o investimento em cenários mais exigentes. Já o Bizagi e o Bonita BPM oferecem vantagens distintas nomeadamente o Bizagi pela sua abordagem low-code e rapidez de implementação e o Bonita pela flexibilidade do modelo *open-source*. Ainda assim, ambos apresentam limitações relevantes quando comparados com o Camunda, sobretudo ao nível da escalabilidade, suporte a regras de decisão e independência tecnológica.

Em síntese, a decisão sobre a plataforma a adotar deve ser baseada numa leitura crítica das necessidades da organização e na capacidade de cada solução se adaptar e evoluir dentro do seu ecossistema tecnológico.

3. Estudos Exploratórios da Plataforma Camunda

Este capítulo apresenta o conjunto de atividades desenvolvidas com o objetivo de explorar e compreender o funcionamento da plataforma Camunda. Para tal, foi utilizado o processo de requerimentos da Escola Superior de Tecnologia e Gestão (ESTG) como caso real, permitindo validar a aplicabilidade da plataforma. As ações realizadas envolveram, numa primeira fase, o levantamento e análise do processo em vigor, seguindo-se a construção de uma prova de conceito com recurso à plataforma Camunda, a respetiva implementação e, por fim, a avaliação do comportamento dos artefactos criados. Todo o trabalho foi conduzido de forma iterativa, permitindo validar gradualmente as funcionalidades da plataforma e estabelecer uma base técnica que poderá ser aprofundada em fases futuras. A descrição inicia-se com a caracterização do processo atual, abordada na próxima subsecção.

3.1. Levantamento e Contextualização do Processo

O levantamento do processo de requerimentos da ESTG foi conduzido através de uma análise da documentação existente e da observação direta dos procedimentos em vigor, complementada por entrevistas não formais com carácter exploratório, ajustado à natureza preliminar da análise com os responsáveis dos diferentes serviços envolvido. Este processo encontra-se atualmente implementado numa plataforma monolítica, apresentando limitações significativas em termos de flexibilidade, manutenção e capacidade de adaptação a novos requisitos da organização.

A análise permitiu identificar três categorias principais de requerentes: estudantes, que submetem pedidos maioritariamente relacionados com questões académicas e dirigidos preferencialmente aos Serviços Académicos ou à Presidência da ESTG; docentes e funcionários não docentes, cujos requerimentos apresentam maior diversidade temática e podem ser dirigidos tanto à Presidência da ESTG como à Presidência do Politécnico do Porto. Para além dos requerentes, foram identificados outros atores, nomeadamente os responsáveis pela triagem nos diferentes serviços e os decisores ao nível da presidência, que assumem papéis distintos na tramitação dos pedidos. Nesta primeira versão do modelo BPMN, optou-se por não incluir o fluxo específico dos Serviços Centrais, uma vez que este engloba um funcionamento organizacional distinto que ultrapassa o âmbito da ESTG e será objeto de desenvolvimento futuro. A partir da análise do funcionamento atual, foi possível estruturar o modelo de processo em quatro *lanes,* sendo essa segmentação posteriormente refletida no modelo BPMN da prova de conceito na *Figura 3*, facilitando a separação lógica das responsabilidades.

3.2. Desenvolvimento da Prova de Conceito

A plataforma Camunda foi utilizada na presente investigação, tendo como objetivo primário o estudo e validação de algumas funcionalidades oferecidas por esta tecnologia, desde a modelação BPMN até ao desenvolvimento de formulários. Este exercício exploratório não visou a migração do sistema existente, mas sim a compreensão aprofundada das capacidades e limitações da plataforma para futura tomada de decisão.

O estudo centrou-se na avaliação de várias componentes tecnológicas da *stack* Camunda. Em primeiro lugar, foi explorado o Camunda *Modeler* para modelação BPMN, tanto na versão *desktop* como na versão *cloud*, ferramentas que se revelaram intuitivas e flexíveis na criação de diagramas. A facilidade de criação e modificação dos artefactos dos processos constituiu um dos aspetos mais positivos desta exploração. O ambiente gráfico permitiu o desenvolvimento rápido do diagrama de processo, com possibilidade de iteração constante. O sistema de *drag-and-drop* para colocação de elementos BPMN, a validação automática da sintaxe que alerta para inconsistências no modelo, e as sugestões contextuais que orientam o utilizador na seleção de elementos apropriados para cada situação. As funcionalidades como a auto conexão de elementos, a criação automática de *sequence flows*, e a capacidade de duplicar e reutilizar componentes do processo torna o desenvolvimento mais rápido. O Camunda *Modeler* oferece ainda assistentes para configuração, como expressões condicionais e mapeamento de dados, reduzindo significativamente a curva de aprendizagem para utilizadores menos familiarizados com BPMN.

A versão cloud apresenta vantagens adicionais em termos de colaboração, permitindo partilha imediata de modelos e trabalho colaborativo em tempo real, funcionalidades particularmente relevantes em contextos organizacionais onde múltiplos *stakeholders* necessitam de contribuir para a definição de processos. A sincronização automática entre versões e o controlo de versões integrado facilitam a gestão de alterações e a manutenção do histórico de evolução dos modelos.

Após a modelação, diagramas BPMN foram implementados no motor *Zeebe* através do Camunda *Modeler*. Este processo de *deployment* regista o modelo no motor, criando uma definição de processo com identificador e versão únicos. O modelo passa assim a estar disponível para a criação de instâncias executáveis, permitindo ao *Zeebe* interpretar e gerir a lógica definida. Durante a execução, o motor avaliou expressões condicionais, encaminhou o fluxo com base em variáveis de entrada e coordenou

ramos paralelos e exclusivos, de acordo com a progressão de cada instância no *workflow*. As ferramentas Camunda *Operate* e *Tasklist* foram igualmente objeto de estudo após o *deployment* dos processos. O *Operate*, utilizado para monitorização e análise do comportamento em tempo de execução, permite observar o estado das instâncias de processo. Através da visualização gráfica do fluxo de execução, é possível identificar tarefas ativas, concluídas ou com falha, bem como aceder às variáveis e registos de erro associados. Esta informação permite apoiar o diagnóstico de bloqueios e a avaliação de tempos de execução por tarefa, fornecendo base para a identificação manual de potenciais *bottlenecks.*

Complementarmente, o *Tasklist* disponibilizou uma interface orientada à execução de *user tasks*, o que permite os utilizadores interagir com os formulários associados e fazer avançar o processo nas suas diferentes fases. As *user tasks* foram configuradas com formulários embebidos através do sistema Camunda Forms, o qual permite definir estruturas de recolha de dados em formato JavaScript Object Notation (JSON) integradas diretamente nas tarefas do processo. Durante os testes, os formulários foram utilizados para suportar a interação dos utilizadores com o fluxo, assegurando a introdução e tratamento de informação necessária em cada ponto. Esta integração viabilizou a simulação dos processos com os diferentes atores simulados.

3.3. Análise do Modelo Implementado e Teste dos Artefactos

O modelo de processo desenvolvido reflete a estrutura lógica e funcional observada no fluxo atual de tramitação de requerimentos, incluindo múltiplos caminhos de decisão, *subprocessos* reutilizáveis e *user tasks* suportadas por artefactos configuráveis. A versão implementada, ilustrada na Figura 3 (diagrama BPMN criado no Camunda Modeler), possibilitou a validação do comportamento dinâmico do processo.

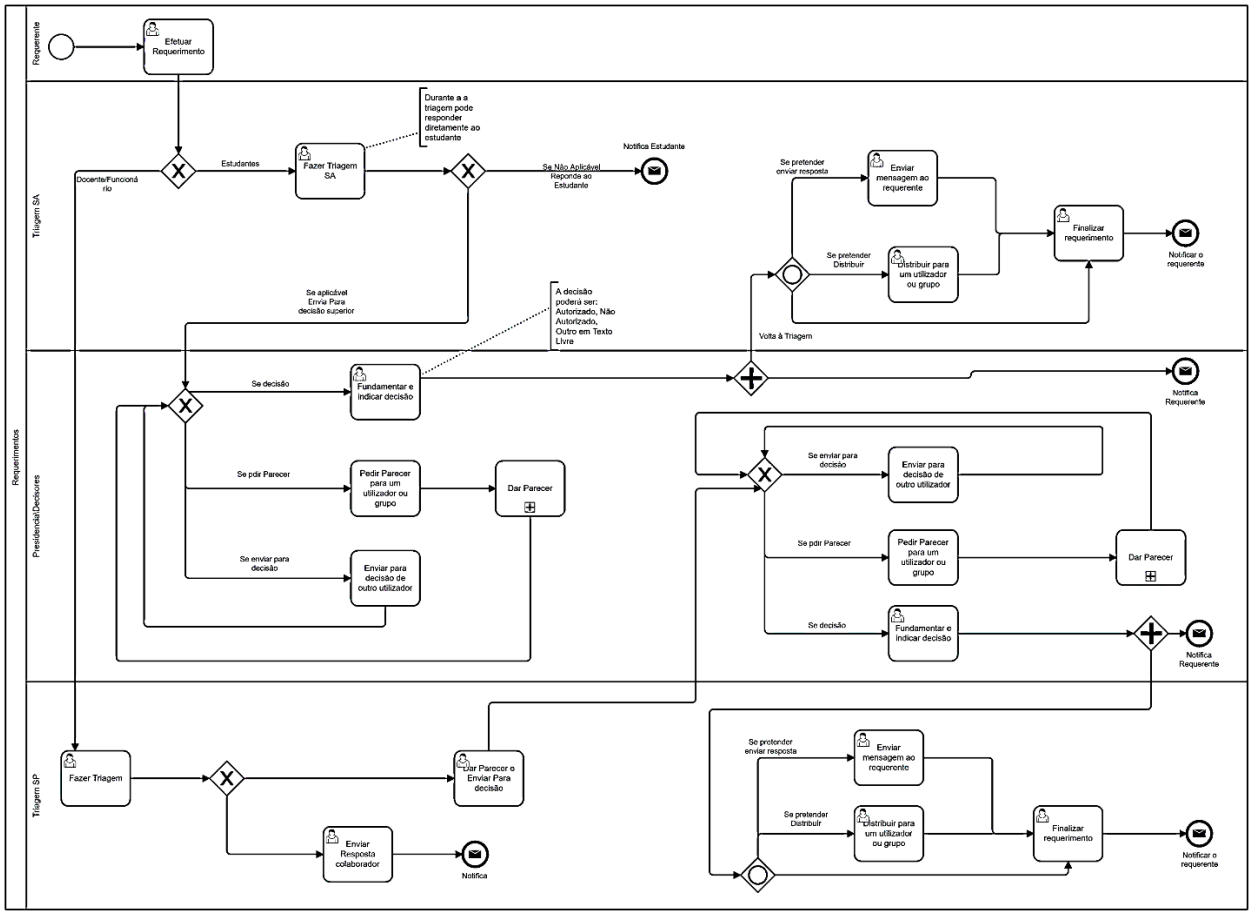


Figura 3 - Modelo BPMN do processo de requerimentos

O fluxo tem início com a submissão do pedido por parte do requerente, sendo imediatamente avaliado com base no perfil do utilizador identificado. Os requerimentos submetidos por estudantes são automaticamente encaminhados para a triagem dos Serviços Académicos, enquanto os de docentes e funcionários não docentes seguem para a triagem da Presidência. Esta diferenciação é implementada através de um *exclusive gateway*, que direciona o percurso da instância conforme as variáveis definidas por tipo de utilizador no momento da submissão. Em ambos os casos, a triagem constitui um ponto de bifurcação em que o pedido pode ser encerrado com resposta direta ou encaminhado para a fase de decisão, consoante a natureza do assunto. A decisão, modelada como uma tarefa exclusiva, pode ser tomada diretamente, delegada ou precedida por pedidos de parecer. A solicitação de parecer ativa um *subprocesso* reutilizável, que pode ser invocado quantas vezes forem necessárias para suportar a fundamentação da decisão. Após a decisão ser formalizada, o processo regressa à respetiva triagem, que comunica o resultado ao requerente e encerra o pedido.

A configuração do modelo garante que todas as decisões são intermediadas pelos serviços responsáveis, respeitando os fluxos formais definidos e mantendo a separação entre triagem e

decisão. A definição dos percursos foi realizada com base na utilização de *exclusive gateways*, que condicionam o encaminhamento da instância de processo em função das variáveis recolhidas na submissão e durante as *user tasks*. A modelação contemplou também a utilização de um *parallel gateway*, aplicado em pontos do processo em que diferentes tarefas podiam decorrer de forma independente, como por exemplo a execução simultânea de notificações. Esta abortdagem possibilita o aumento da flexibilidade do processo e melhora a capacidade de adaptação a cenários com múltiplos intervenientes. Para além disso, foram utilizados artefactos de definição de variáveis e mapeamentos de entrada e saída, bem como tarefas de serviço para simular pontos de integração ou ações automatizadas, permitindo testar a consistência das execuções e o comportamento dinâmico dos diferentes elementos modelados.

As *user tasks* incluídas no modelo foram executadas com o apoio de formulários embebidos, que serviram de interface para a introdução e manipulação de dados. A Figura 4 exemplifica uma tarefa de triagem realizada pelos Serviços Académicos, onde o operador pode consultar os dados submetidos pelo requerente, selecionar o encaminhamento apropriado e introduzir comentários. Estes formulários foram configurados com o *Camunda Forms* e disponibilizados diretamente no *Tasklist*, o que permite simular a execução do processo.

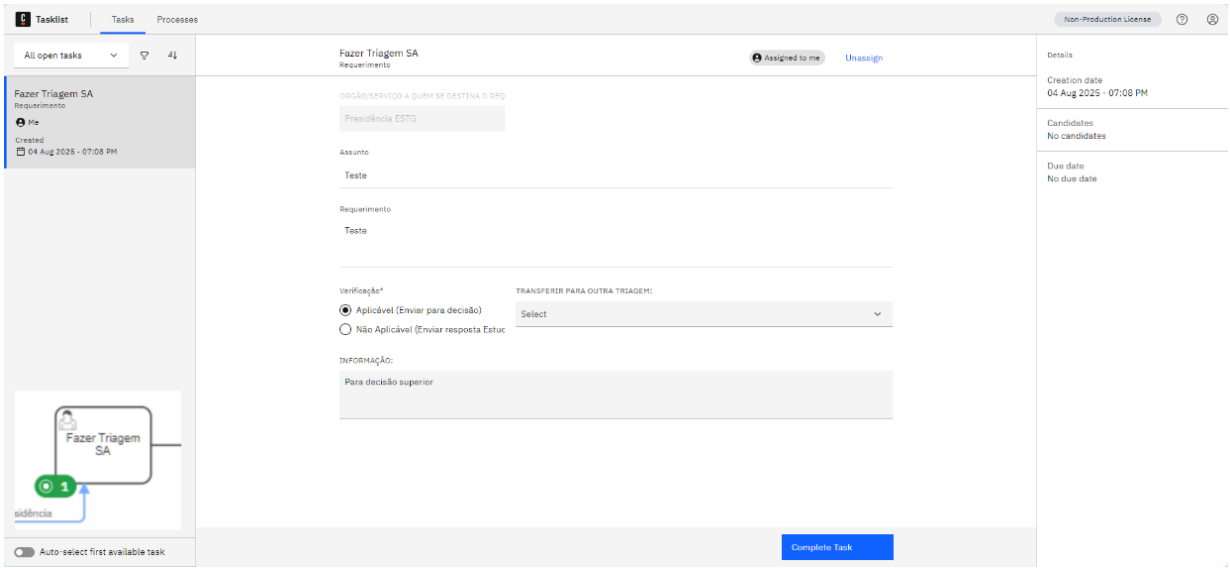


Figura 4 - Formulário de triagem

A Figura 5 apresenta a tarefa de decisão/despacho, na qual o utilizador com responsabilidades de decisão pode visualizar todo o conteúdo do requerimento e a respetiva informação da triagem, selecionar o tipo de despacho (autorizado, não autorizado ou não aplicável), introduzir a

fundamentação e, se necessário, solicitar pareceres adicionais, delegar a decisão ou redirecionar o processo para outras áreas de triagem. O *Tasklist* permite assim a execução sequencial das tarefas atribuídas, controlando a lógica de fluxo com base nas opções selecionadas e nas variáveis de contexto.

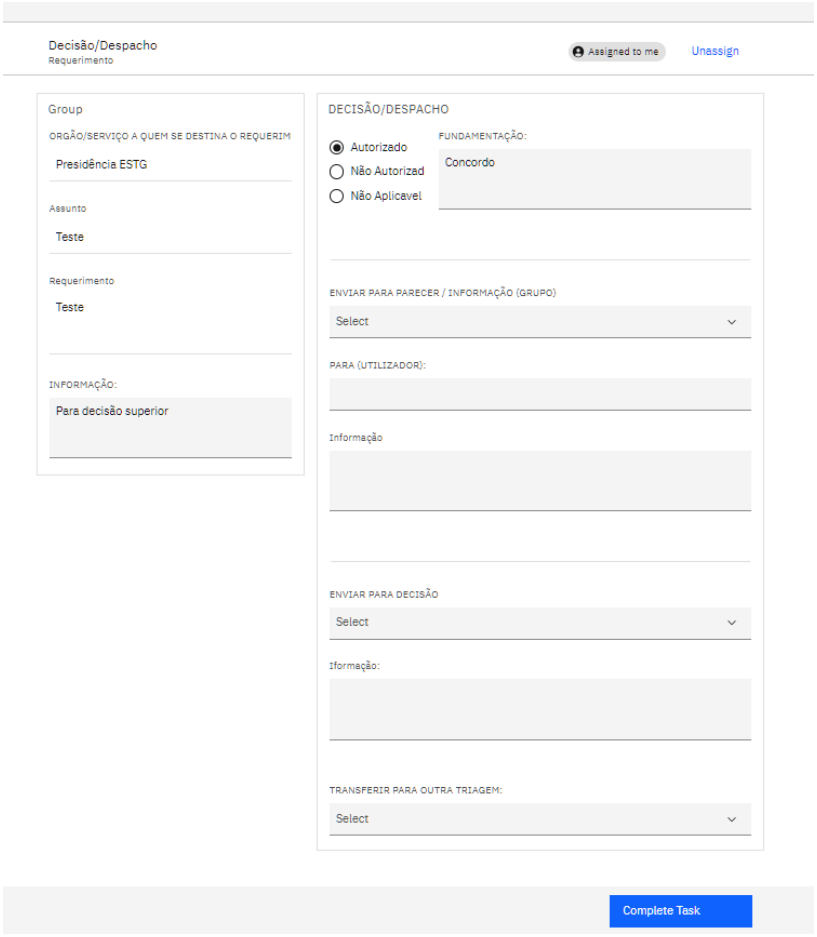


Figura 5 - Formulário despacho/decisão

A Figura 6 ilustra o acompanhamento da instância de processo no *Camunda Operate*, onde é possível visualizar em tempo real os elementos já percorridos, o estado atual do processo e os próximos passos a executar. A interface fornece também acesso direto às variáveis de processo, permitindo a sua consulta e a própria edição manual para testes. Esta funcionalidade possibilita a verificação do comportamento de cada instância, a simulação de diferentes caminhos do processo e a deteção de potenciais inconsistências. O *Operate* fornece ainda informações sobre falhas de execução, tarefas pendentes, tempos de execução e estrutura das instâncias ativas, permitindo o diagnóstico do ciclo de vida de cada execução.

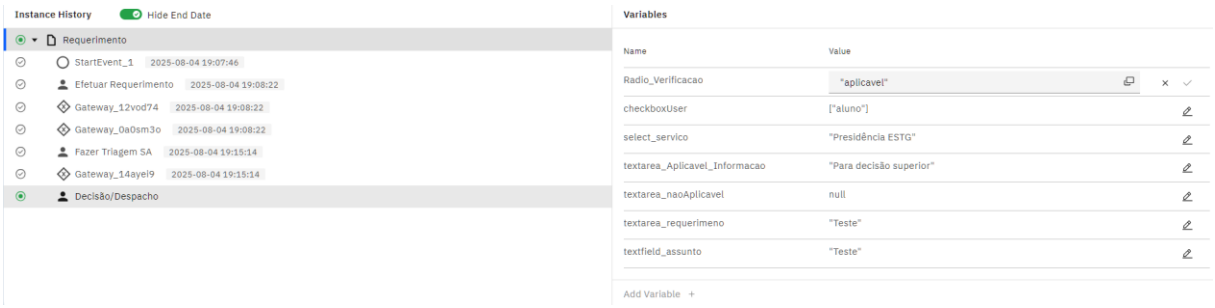


Figura 6 - Instância de processo no Camunda Operate

Como se pode verificar, o processo adota uma arquitetura modular, com utilização de expressões condicionais e componentes reutilizáveis. Esta configuração não só demonstra a aplicabilidade do modelo a diferentes cenários, como também constitui uma base de validação técnica do trabalho, ao permitir: confirmar a correta orquestração das tarefas humanas e automáticas; integração e fluxo das variáveis de processo; a monitorização das instâncias de processo no *Operate*; e o funcionamento da lógica de decisão no motor de execução do Camunda.

Importa, contudo, sublinhar que esta implementação corresponde a uma primeira iteração no contexto de um estudo exploratório. De acordo com o ciclo de vida da gestão de processos de negócio, abordado no capítulo de revisão do estado da arte, será necessária a realização de várias interações adicionais em fases futuras, com validações incrementais e melhorias do processo. Esta versão inicial permitiu apenas testar os princípios fundamentais de modelação e execução, sem abranger ainda a totalidade da complexidade organizacional envolvida.

3.4. Considerações Técnicas e Conclusões do Estudo Exploratório

A prova de conceito permitiu validar, de forma segmentada, as principais funcionalidades oferecidas pela *stack* Camunda. O ambiente de execução foi instanciado localmente em ambiente de desenvolvimento. Para isso foi utilizado o *Docker Compose* com as imagens oficiais disponibilizadas pela plataforma, assegurando a replicabilidade e o isolamento do contexto de teste.

Apesar de não terem sido alvo de análise exaustiva nesta fase, os principais componentes da *stack* Camunda demonstraram um comportamento estável e coerente com a documentação técnica e com o estudo apresentado no capítulo anterior. Esta constatação foi aferida através da execução de testes funcionais sobre o modelo implementado, verificou-se a correta interpretação das *user tasks*, das transições condicionais e dos eventos definidos no diagrama BPMN. Os testes foram realizados no Camunda *Tasklist*, permitindo simular a execução completa das tarefas atribuídas, e analisados no

Camunda *Operate*, onde se confirmou o fluxo das instâncias e a consistência das variáveis de processo. O motor de execução revelou-se fiável na orquestração das instâncias e no cumprimento do comportamento previsto em todos os cenários testados.

O sistema de formulários do Camunda 8 mostrou-se adequado para cenários básicos de recolha de dados, garantindo a integração com as *user tasks* e a execução de fluxos de trabalho. Todavia, a construção de interfaces mais complexas apresentou limitações técnicas consideráveis. Apesar da geração automática da estrutura através do *Camunda Forms*, a personalização avançada exige conhecimentos técnicos e integração com código adicional. A execução de ações específicas dentro dos formulários obriga à definição de *Script Functions* em JavaScript, bem como à utilização de *input/output mappings* e expressões técnicas, o que reduz a acessibilidade da ferramenta a utilizadores não técnicos.

O *Camunda Forms* não oferece suporte nativo para lógica condicional dinâmica entre campos (como mostrar/ocultar secções baseado em seleções anteriores) nem para interação avançada em tempo real (validação instantânea, *autocomplete* com API externas, ou cálculos automáticos). Para implementar estas funcionalidades, é necessário abandonar o sistema de formulários integrado e desenvolver interfaces completamente personalizadas utilizando a *Camunda Tasklist API* em conjunto com HTML, JavaScript ou *framework s frontend* dedicadas. Estas exigências técnicas reforçam a conclusão apresentada na análise comparativa na secção 2.5.4, segundo a qual a abordagem *low-code* da plataforma revela-se limitada. Esta limitação sugere a necessidade de avaliar soluções híbridas que combinem o motor de *workflow* Camunda com tecnologias *frontend* mais flexíveis para a criação de formulários sem tantas limitações.

4. Metodologia/Desenho da Solução e Tecnologias

Dando seguimento ao estudo exploratório e aos testes realizados no capítulo anterior, que permitiram compreender o funcionamento da plataforma Camunda e identificar restrições inerentes à abordagem *low-code*, o presente capítulo descreve a metodologia adotada e o desenho da proposta para a automatização do processo de autorização de condução de viatura da ESTG.

Com base nas conclusões obtidas na fase experimental, foi definida uma abordagem híbrida, que combina o motor de *workflow* Camunda com uma arquitetura modular baseada em microserviços e componentes personalizados**,** superando as restrições observadas durante a fase exploratória. O capítulo apresenta os padrões da arquitetura, as tecnologias selecionadas, a análise do processo atual e a arquitetura proposta**,** evidenciando como a solução final concretiza os princípios de automatização e integração definidos na fase de estudo exploratório.

4.1. Padrões de Desenvolvimento em Arquitetura de Microserviços

A implementação de arquiteturas de microserviços requer padrões que abordem os desafios de sistemas distribuídos, nomeadamente a separação de responsabilidades, comunicação entre serviços, gestão de dados, segurança e observabilidade. Estes padrões fornecem soluções para construção de sistemas onde cada microserviço evolui independentemente mantendo coerência global.

Esta secção descreve os padrões de arquitetura que orientaram o desenvolvimento da solução, desde a modelação do domínio até às estratégias de comunicação e descoberta de serviços.

4.1.1. *Domain-Driven Design* Simplificado

O *Domain-Driven Design* (DDD) é uma abordagem de arquitetura que coloca o domínio do negócio no centro do desenvolvimento de software, o que promove que os conceitos e regras da organização se refletem diretamente no modelo implementado. A sua aplicação procura reduzir a distância entre especialistas funcionais e equipas técnicas, através do uso de uma linguagem comum que assegura consistência semântica em todas as fases do desenvolvimento. Em arquiteturas de microserviços, o DDD possibilita a decomposição do sistema em *bounded contexts*, ou seja, módulos autónomos que encapsulam responsabilidades bem delimitadas do domínio. Esta prática não só facilita a manutenção e a escalabilidade, como também promove maior coesão interna e reduz a dependência entre microserviços. Evidências recentes demonstram que a utilização do DDD, mesmo em versões

simplificadas, contribui para uma melhor gestão da complexidade o que permite alinhar de forma mais próxima o modelo conceptual do negócio com a sua implementação técnica [60].
No contexto do presente projeto, foi implementada uma abordagem simplificada do *Domain-Driven Design* (DDD) para orientar a definição das fronteiras dos microserviços e a organização da lógica de domínio. A estratégia adaptou os conceitos fundamentais do DDD às limitações existentes, preservando os benefícios de separação de responsabilidades, modelação orientada ao domínio e coesão interna. Esta simplificação permite equilibrar rigor e reduzir complexidade, enquanto melhora a testabilidade e assegura que o modelo conceptual do negócio se mantém alinhado com a implementação técnica, refletindo de forma eficaz as novas responsabilidades atribuídas ao sistema à medida que os requisitos evoluem.

4.1.2. Circuit Breaker Pattern

O *Circuit Breaker* é um padrão de conceção que previne a execução de operações com elevada probabilidade de falha, evitando *timeouts* e o consumo desnecessário de recursos. Este padrão monitoriza continuamente as falhas e, quando um limiar é atingido, abre o circuito, o que faz com que as chamadas subsequentes falhem imediatamente durante um período determinado. Em arquiteturas de microserviços, o *Circuit Breaker* impede que a indisponibilidade de um serviço dependente cause falhas em cascata [61].

4.1.3. API Gateway Pattern

O *API Gateway Pattern* é um padrão de arquitetura que centraliza o acesso a múltiplos microserviços através de um ponto único de entrada, que resolve problemas fundamentais em arquiteturas distribuídas, nomeadamente proliferação de *endpoints*, gestão fragmentada de políticas de segurança, controlo de tráfego disperso e a dependência direta entre aplicações cliente e serviços *backend*. Este padrão abstrai a complexidade da comunicação com múltiplos serviços e oferece interface unificada que simplifica a interação com sistemas distribuídos [62].

Comparativamente à comunicação direta com cada microserviço, esta implementação elimina a necessidade de conhecimento distribuído de *endpoints* e protocolos específicos. O padrão permite a agregação de respostas quando necessário e salienta o papel da centralização na simplificação de arquiteturas distribuídas e na manutenção da flexibilidade operacional.

4.1.4. Event-Driven Pattern

O *Event-Driven Pattern* é um padrão de arquitetura que promove comunicação assíncrona entre microserviços através da publicação e subscrição de eventos, eliminando dependências diretas e temporais entre os serviços. Este padrão resolve problemas de consistência de dados em transações, onde a comunicação síncrona pode causar falhas em cascata e reduzir a disponibilidade global. O padrão facilita escalabilidade, tolerância a falhas e permite que os serviços evoluam independentemente[36].

4.1.5. Health Check Pattern

O *Health Check Pattern* é um padrão de observabilidade em arquiteturas de microserviços que normalmente é implementado por um *endpoint* (tipicamente HTTP/*health*) para verificar o estado dos microserviços. Este padrão resolve o problema de detetar quando uma instância pode já não estar apta a processar pedidos, embora continue em execução, uma situação típica em sistemas distribuídos, onde falhas parciais podem ocorrer sem sinais evidentes. O *endpoint* executa verificações que incluem o estado das ligações com serviços de infraestrutura, disponibilidade de recursos do sistema e validações da lógica de negócio. A implementação permite verificações periódicas para determinar a disponibilidade da instância o que facilita mecanismos de recuperação e integração com plataformas de orquestração [36].

4.1.6. Database per Service Pattern

O *Database per Service Pattern* é um padrão de arquitetura de dados que estabelece que cada microserviço deve possuir a sua própria base de dados privada, acessível apenas através da sua API. Este padrão resolve problemas de dependência de dados em sistemas distribuídos, onde múltiplos serviços acedem a bases de dados partilhadas, o que limita a capacidade de desenvolvimento, implementação e escalabilidade independente dos serviços. O padrão garante autonomia de dados, permite escolha de tecnologias de armazenamento adequadas às necessidades específicas de cada serviço e facilita escalabilidade horizontal independente [36].

Na implementação deste trabalho, o padrão foi aplicado considerando as necessidades de persistência específicas de cada microserviço. A estratégia adotada garante isolamento, permitindo que cada serviço aceda apenas aos seus próprios dados através de *framework* s como o Entity *Framework* . Desta forma, preserva-se a autonomia, evitam-se dependências diretas entre bases de dados e cada instância

pode ser dimensionada de forma independente. Além disso, problemas num serviço não afetam os restantes e mantém-se a flexibilidade para futuras migrações para diferentes tecnologias de armazenamento.

4.1.7. Access Token Pattern

O *Access Token Pattern* permite propagação de identidade entre microserviços através de *tokens* que encapsulam dados do utilizador e permissões. Richardson, em [36], descreve este padrão como solução para autenticação distribuída onde cada serviço verifica a identidade do utilizador de forma independente. Tokens JSON Web Token (JWT) são *self-contained*, incluindo toda a informação necessária sobre o utilizador e as suas permissões, verificáveis através de assinatura criptográfica sem necessidade de chamadas síncronas a serviços externos. Por ser *self-contained* e assinado criptograficamente, o JWT elimina a necessidade de chamadas síncronas a serviços externos para verificação, permitindo uma autorização granular sem manter estado de sessão.

Neste trabalho, o padrão foi implementado com *JavaScript Web Tokens* emitidos pelo *Duende IdentityServer*. A implementação desta solução estabelece uma estrutura de *scopes* granulares que permite validação distribuída, onde cada microserviço mantém autonomia na verificação de autorização sem comprometer a segurança ou criar dependências síncronas entre serviços. Os detalhes técnicos da implementação são apresentados na secção 5.1.1.2 Implementação do *Access Token Pattern*

4.1.8. Service Registry Pattern

O *Service Registry Pattern* é um padrão de arquitetura que permite descoberta automática de serviços em ambientes distribuídos, resolvendo problemas de localização dinâmica onde os serviços podem ser implementados em diferentes endereços IP e portas. Este padrão facilita escalabilidade horizontal, tolerância a falhas através de registos centralizados ou mecanismos nativos das plataformas de orquestração [36]. Enquanto tradicionalmente era necessário implementar e gerir manualmente sistemas como Eureka, Consul ou Netflix OSS, atualmente as plataformas de orquestração de *containers* fornecem estas tipo de funcionalidades de forma nativa, embora as implementações dedicadas possam oferecer funcionalidades mais avançadas para casos específicos.

Neste trabalho a descoberta automática é assegurada pelos mecanismos nativos das plataformas de orquestração utilizadas. No K3s, esta funcionalidade é dada pelo *CoreDNS* e pelos *objetos Service*,

enquanto no *Docker* é realizada através do DNS interno das redes definidas. O acesso externo aos serviços no K3s é efetuado através de um *Ingress (Traefik),* que expõe *endpoints* específicos. A Figura 7 mostra a estrutura de descoberta, com dois domínios independentes: K3s e *Docker*.

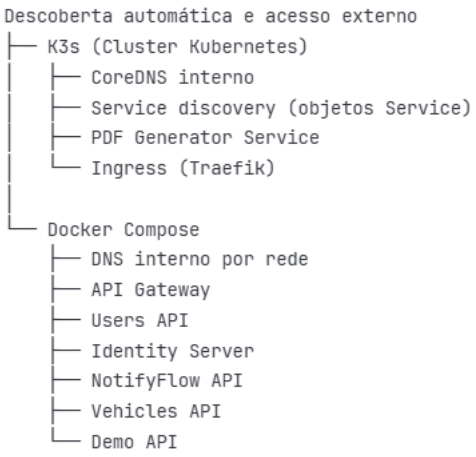


Figura 7 - Modelo de service registry

A implementação demonstra como diferentes tecnologias fornecem descoberta automática sem necessidade de configuração manual. Os serviços tornam-se acessíveis através de DNS em vez de endereços IP fixos o que simplifica os *redeployments* e cria base para futuras evoluções, como mecanismos de registo e balanceamento de carga.

4.2. Deployment Blue-Green Pattern

Os padrões de *deployment* abordam os desafios de atualização de múltiplos serviços independentes, focando na minimização de tempo de indisponibilidade e redução de riscos durante atualizações do sistema.

O *Blue-Green Deployment* é um padrão de implementação que reduz o *downtime* e o risco durante as atualizações de aplicações através da manutenção de dois ambientes de produção idênticos. Apenas um dos ambientes (*Blue* ou *Green*) serve o tráfego de produção num dado momento, enquanto o outro permanece inativo até ser necessário para uma nova versão. Este padrão permite testes completos da nova versão antes da transição e oferece capacidade de *rollback* imediato em caso de falhas [63].

4.3. Principais Tecnologias Utilizadas no Caso de Estudo

A arquitetura da solução proposta incorpora tecnologias selecionadas segundo critérios técnicos como compatibilidade com padrões abertos, modularidade, maturidade, integração facilitada e suporte à automação de processos de negócio. Esta secção apresenta uma descrição das principais tecnologias utilizadas, organizadas por domínio funcional, com a respetiva fundamentação.

4.3.1. Backend

A camada de *backend* da solução integra um conjunto de tecnologias selecionadas com base em critérios técnicos, nomeadamente a compatibilidade com padrões abertos, a capacidade de integração em arquiteturas distribuídas e a disponibilidade de documentação. Estas escolhas foram também influenciadas pelo enquadramento metodológico definido ao longo do desenvolvimento do trabalho, bem como pela experiência acumulada na utilização de algumas dessas ferramentas.

A plataforma base adotada foi o ASP.NET Core, uma *framework open-source* mantida pela Microsoft, orientada ao desenvolvimento de aplicações web multiplataforma. A seleção desta tecnologia teve como base o seu suporte nativo à injeção de dependências, através de um sistema integrado que permite a configuração explícita de serviços com ciclos de vida bem definidos (*Singleton, Scoped e Transient*). Esta abordagem facilita a modularização da lógica de negócio e a testabilidade dos componentes, sem necessidade de bibliotecas externas para gestão de dependências. Adicionalmente, o modelo de *middleware* do *ASP.NET* Core proporciona um *pipeline* HTTP configurável, que permite o tratamento estruturado de pedidos, autenticação, autorização e serialização de dados, de forma flexível e extensível [64].

A camada de acesso a dados foi implementada com recurso ao *Entity Framework Core* (*EF Core*), um *Object-Relational Mapper* (ORM) que simplifica a comunicação com bases de dados relacionais através do mapeamento entre classes de domínio e estruturas relacionais. O *EF Core* disponibiliza um sistema de *migrations* que permite a evolução controlada do esquema da base de dados, assegurando a consistência entre ambientes de desenvolvimento e produção mediante *scripts* com gestão de versões. Para cenários que requerem maior controlo na definição de consultas ou otimização de desempenho, recorreu-se ao micro-ORM NPoco, valorizado por ser leve e flexível na execução de instruções *Structured Query Language* (SQL) parametrizadas. A adoção desta abordagem híbrida permitiu conciliar a produtividade do *EF Core* nas operações CRUD [65]. O mapeamento automático entre entidades de domínio e objetos de transferência de dados (DTO) foi realizado com a biblioteca

AutoMapper, promovendo a separação de responsabilidades na lógica das aplicações. A base de dados relacional adotada foi o MySQL, acedida através do *provider* Pomelo.Entity*Framework* Core.MySql, integrado no ecossistema .NET. Esta opção assegura a compatibilidade com o *EF Core* e facilita a gestão estruturada e consistente dos dados aplicacionais.

O mecanismo de autenticação da solução foi estruturado com base em *ASP.NET Core* Identity, um sistema modular para gestão de utilizadores, autenticação e controlo de acessos. Esta abordagem foi complementada pela integração com o Duende IdentityServer, que implementa protocolos de segurança padronizados como *OAuth* 2.0 e OpenID Connect, possibilitando comunicação federada entre os diversos componentes da arquitetura. A escolha do Duende IdentityServer, em detrimento de soluções externas como o Keycloak, fundamentou-se na sua integração nativa com o ecossistema .NET, facilitando a configuração, manutenção e extensão, bem como na capacidade de personalização de políticas de autenticação de acordo com os requisitos específicos do sistema. A solução implementada assegura o suporte a protocolos de segurança nas comunicações entre serviços, encriptação de dados sensíveis e mecanismos de gestão de identidade, como *Single Sign-On (SSO)*, controlo de acesso baseado em *roles* (RBAC) e gestão de sessões distribuídas. Esta integração garante controlo granular sobre *tokens*, políticas de autorização e auditoria de acessos, promovendo a interoperabilidade segura com aplicações externas e mantendo a coerência tecnológica da arquitetura [66].

A proteção de informação sensível, como *tokens* de acesso, *cookies* de autenticação e dados de sessão, foi garantida através da biblioteca Microsoft.AspNetCore.DataProtection, que implementa algoritmos de cifra, com suporte a rotação automática de chaves e armazenamento seguro. O sistema recorre a JSON Web Tokens (JWT) como formato de autenticação *stateless*, permitindo que os pedidos às API sejam validados sem necessidade de consulta constante à base de dados. Esta abordagem melhora o desempenho em arquiteturas distribuídas, reduz a latência das operações autenticadas e facilita a escalabilidade horizontal. Os tokens JWT são bastantes utilizados como standard na autenticação entre sistemas distribuídos e API REST, justificando a sua aplicação neste contexto [67]

A comunicação assíncrona entre componentes foi implementada com recurso ao RabbitMQ, um *message broker open-source* que suporta o protocolo *AMQP* (*Advanced Message Queuing Protocol*), assegurando a entrega de mensagens através de mecanismos como persistência e confirmação (*acknowledgment*). O RabbitMQ disponibiliza suporte para vários padrões de *messaging*, incluindo *publish-subscribe*, *request-reply* e *work queues*, e proporciona flexibilidade na interação entre

microserviços. A sua utilização permite o desacoplamento temporal e lógico entre serviços, contribuindo para a tolerância a falhas através de funcionalidades como *dead-letter queues*, políticas de reenvio e *clustering* para alta disponibilidade. Esta abordagem baseada em eventos reforça a resiliência, escalabilidade horizontal e capacidade de evolução da arquitetura, permitindo que os serviços operem de forma autónoma [68].

O encaminhamento de pedidos externos foi centralizado num *API Gateway* desenvolvido com a biblioteca *Ocelot*, que implementa mecanismos de encaminhamento com base em padrões de *Uniform Resource Locator* (URL), cabeçalhos HTTP e métodos de requisição. A opção pelo *Ocelot*, em detrimento de soluções como *Kong*, *Zuul* ou *AWS API Gateway*, foi motivada pela sua integração nativa com o ecossistema .NET, compatibilidade com o *middleware pipeline* do *ASP.NET Core* e configuração declarativa através de ficheiros *JSON*. A biblioteca disponibiliza funcionalidades como *rate limiting* para controlo de tráfego, *circuit breaker* para gestão de falhas, transformação de pedidos e respostas, *caching* distribuído e balanceamento de carga com suporte a algoritmos *round-robin* e *least connections*. O sistema de encaminhamento permite configuração dinâmica, com redirecionamento baseado em critérios como controlo de versões de API, geolocalização e *service discovery*. Esta camada atua como ponto único de entrada (*single entry point*), mediando a comunicação entre clientes externos e microserviços internos através de técnicas de *reverse proxy*, com validação centralizada de *tokens*, aplicação de políticas *CORS* e ocultação da topologia da rede. O *Ocelot* permite *debugging* no *Visual Studio* e alinhamento tecnológico com os restantes componentes da solução, facilitando a orquestração de serviços distribuídos e a monitorização através de *logging [69].*

A documentação das API foi gerada automaticamente com recurso à biblioteca *Swashbuckle.AspNetCore*, que suporta a *OpenAPI Specification* (anteriormente parte do projeto *Swagger*). Esta biblioteca integra-se com o ASP.NET Core e permite a geração de documentação interativa, estruturada, conforme os padrões atuais [67].

Por fim, foram integradas bibliotecas auxiliares como a MailKit para o envio seguro de e-mails através de protocolos SMTP, a iText7 para geração dinâmica de documentos PDF com formatação personalizável, e os pacotes *System.DirectoryServices* e *System.DirectoryServices.Protocols* para integração com o diretório LDAP interno da organização. A implementação LDAP permite a autenticação centralizada dos utilizadores através do *Active Directory*, eliminando a necessidade de gestão duplicada de credenciais e garantindo consistência com as políticas de segurança existentes. Esta integração suporta operações de *bind*, consulta de atributos de utilizador e validação de grupos

de acesso, proporcionando um mecanismo de *Single Sign-On* (SSO) que se alinha com a infraestrutura de diretório já estabelecida na instituição.

4.3.2. Frontend

A camada de *frontend* foi desenvolvida com o objetivo de suportar a interação entre o utilizador e os serviços expostos pela aplicação. A seleção das tecnologias teve como base a compatibilidade com API REST, suporte a aplicações *single-page* e integração com mecanismos de autenticação bem como experiência acumulada na utilização de algumas dessas ferramentas.

A interface foi construída com Angular, uma *framework* baseada em *TypeScript* que disponibiliza componentes reutilizáveis, injeção de dependências e um sistema de modularização[70]. A escolha do Angular, face a alternativas como React ou Vue.js, fundamentou-se na sua integração nativa de funcionalidades, como *routing*, cliente HTTP e gestão de estados, reduzindo a dependência de bibliotecas externas. A estruturação em módulos promove separação clara de responsabilidades, facilitando a manutenção em ambientes distribuídos. O suporte a TypeScript assegura tipagem estática e deteção antecipada de erros em tempo de compilação. Para o tratamento de eventos assíncronos e manipulação de dados reativos, foi utilizada a biblioteca *RxJS*. Esta ferramenta permite a criação de fluxos de dados baseados em observables, o que facilita a comunicação entre serviços e a sincronização de estados através de operadores como map, filter e switchMap.

Para automatizar o envio de credenciais nas requisições, foi implementado um HttpInterceptor responsável por incluir o *token JSON Web Token* (JWT) no cabeçalho *Authorization* de cada chamada HTTP. Esta abordagem permite autenticação *stateless* baseada em *token*, eliminando a necessidade de gestão de sessões no cliente.

A biblioteca jwt-decode foi utilizada para extrair e interpretar o conteúdo dos *tokens* JWT no *browser*, permitindo obter dados como o identificador do utilizador, perfis de autorização e tempo de expiração, o que possibilita a gestão automática da renovação dos *tokens*.

Foram integradas bibliotecas complementares para funcionalidades específicas:

- **SweetAlert2**: apresentação de caixas de diálogo com suporte a interações de confirmação e notificações;
- **ngx-pagination**: gestão da paginação de listas com controlo de página e tamanho;
- **bpmn-js**: renderização e manipulação de diagramas BPMN no navegador.

A camada de apresentação foi implementada com o *Tailwind CSS* como *framework* de estilos principal, tendo o *Bootstrap* sido utilizado para determinados componentes da interface. A exibição de ícones foi assegurada através da biblioteca *Font Awesome*. O ambiente de desenvolvimento foi configurado com o *Prettier* para formatação automática do código e o *ESLint* para validação e aplicação de boas práticas de programação.

4.3.3. Infraestrutura e Orquestração

A infraestrutura foi implementada com recurso a tecnologias de containerização, com o objetivo de garantir isolamento entre serviços, replicação controlada de ambientes e compatibilidade entre sistemas heterogéneos. As tecnologias utilizadas foram selecionadas com base em critérios de portabilidade, adesão a padrões abertos e integração com práticas comuns de desenvolvimento e operação.

A execução dos serviços foi suportada por *Docker*, uma plataforma de *containers* que permite o encapsulamento de aplicações e respetivas dependências em imagens reutilizáveis. O *Docker* permite o controlo de versões das imagens, automatizar a construção de ambientes, manter consistência entre diferentes etapas do ciclo de desenvolvimento e facilita a migração entre diferentes máquinas e sistemas. A orquestração local dos containers foi realizada com *Docker Compose*, ferramenta que permite definir, através de ficheiros YAML, os serviços, configurações, variáveis de ambiente, redes e volumes necessários. Esta abordagem simplifica o arranque conjunto dos serviços e facilita a configuração de ambientes de teste rapidamente reprodutíveis [71].

A gestão dos ambientes de containers foi realizada com o recurso ao *Portainer*, uma aplicação baseada na web que fornece uma interface gráfica para administração de motores *Docker* locais ou remotos. Através do *Portainer*, é possível criar e configurar *stacks*, gerir volumes e redes, visualizar *logs* em tempo real, monitorizar o estado dos serviços e aplicar ações operacionais básicas como *start*, *stop* e *restart* de *containers*. O uso desta ferramenta permite reduzir a necessidade de interação com a linha de comandos e torna mais acessível a gestão da infraestrutura em ambientes de desenvolvimento, possibilitando monitorizar todos os ambientes *Docker* num só local [72].

Para orquestração de *containers* em ambiente local, foi utilizado o K3s, uma distribuição certificada CNCF do *Kubernetes*, otimizada para execução em sistemas com recursos limitados. O K3s mantém compatibilidade total com a API *Kubernetes* e suporta objetos como *Pod* (unidade mínima de execução que pode conter um ou mais *containers*), *Deployment*, *Service*, *Ingress* e *ConfigMap*, permitindo assim

a reutilização de manifestos YAML *standard*. A escolha do *K3s* em detrimento de uma instalação convencional de *Kubernetes* baseou-se nos seguintes fatores:

- Redução de complexidade operacional: o *K3s* disponibiliza um binário único que integra os principais componentes do plano de controlo (kube-apiserver, controller-manager, scheduler) e mecanismos de execução como *containerd*, que simplifica a instalação e a manutenção do sistema;
- Eficiência na utilização de recursos: avaliações comparativas demonstram que o *K3s* apresenta consumo reduzido de *Central Processing Unit* (CPU)e memória em comparação com outras distribuições, mantendo capacidades funcionais equivalentes;
- Compatibilidade com ambientes distribuídos: o *K3s* suporta as mesmas definições e estruturas de configuração que clusters *Kubernetes* completos, o que permite transição futura para ambientes *cloud-managed* sem necessidade de alterações estruturais nos manifestos [73].

O K3s oferece assim uma solução equilibrada entre simplicidade e conformidade com standards de orquestração, adequando-se ao contexto de desenvolvimento e prototipagem da solução implementada.

4.3.4. Automação de Processos

A automação de processos foi implementada com o Camunda 8, num ambiente local containerizado através de *Docker Compose*. Foram incluídos os componentes principais: *Zeebe* (motor de processos), *Operate* (monitorização), *Tasklist* (gestão de *user tasks*), *Optimize* (análise de desempenho), Identity e Keycloak (gestão de autenticação e autorização). Esta configuração permitiu validar o comportamento da solução de forma modular e realista, sem dependência de infraestruturas externas.

A escolha do Camunda 8 resultou da análise realizada no Capítulo 2.5 a diferentes soluções de mercado, na qual se identificaram vantagens e adequação ao contexto face a alternativas como o Bizagi e o Bonita BPM. A decisão foi orientada por critérios como a arquitetura distribuída, o suporte nativo aos *standards* BPMN 2.0 e DMN, a capacidade de integração com sistemas externos e a compatibilidade com ambientes híbridos. Estes fatores, analisados comparativamente, permitiram concluir que o Camunda apresenta uma relação equilibrada entre flexibilidade técnica, escalabilidade e alinhamento com arquiteturas orientadas a microserviços.

Esta abordagem mostrou-se adequada para efeitos de desenvolvimento e testes locais, permitindo a replicação controlada de um ambiente completo da plataforma. A segmentação dos serviços em

containers distintos facilitou a gestão e a compreensão do funcionamento de cada componente. O Camunda 8 revelou-se uma solução tecnicamente adequada, suportando *workflows* distribuídos, monitorização em tempo real e análises de desempenho baseadas em métricas processuais. A possibilidade de operar em cenários *on-premises*, *cloud* ou híbridos reforça a adaptabilidade da plataforma, permitindo a sua evolução futura sem dependência de fornecedores ou infraestruturas específicas.

4.3.5. Monitorização e Análise de Dados

A monitorização do sistema foi realizada com o *Grafana*, aproveitando a experiência prévia com a ferramenta, o seu carácter *open-source* e a capacidade de integração com múltiplas fontes de dados. A plataforma disponibiliza um motor de consultas que suporta agregações, filtros temporais e operações matemáticas sobre séries de dados. Estas capacidades permitem calcular médias de tempo de execução de tarefas, identificar desvios estatísticos no desempenho e acompanhar tendências de carga do sistema. A visualização em tempo real facilita a identificação de gargalos e a otimização de recursos. O sistema de alertas do *Grafana* possibilita a definição de limiares para diferentes métricas, gerando notificações quando valores são ultrapassados, detetando anomalias de desempenho e comportamentos inesperados. A ferramenta suporta diferentes tipos de visualização, desde gráficos de linhas, barras, histogramas e heatmaps, adequando-se a diversos cenários de análise.

O carácter *open-source* do *Grafana* elimina custos de licenciamento, tornando-o adequado para diferentes contextos. A documentação disponível e o suporte da comunidade facilitaram a implementação e configuração da ferramenta [74]

4.3.6. Testes de Software

A implementação de testes constitui uma prática com uma importância relevante para assegurar a fiabilidade e a manutenção da solução.

No *backend*, os testes unitários foram desenvolvidos com a *framework* xUnit, que disponibiliza integração nativa com o *Visual Studio* e suporte à execução via .NET CLI. Utiliza atributos como `Fact` para testes simples e `Theory` para testes parametrizados, permitindo validar diferentes cenários com múltiplos conjuntos de dados. O xUnit suporta *dependency injection* e *mocking*, e facilita o isolamento dos componentes a testar.

Para validação das API, foi utilizado o *Postman* que permite criar *collections* organizadas por funcionalidades e suporta variáveis globais e de ambiente, o que facilita a reutilização de *tokens* de autenticação e a execução em diferentes contextos. As *collections* podem ser armazenadas localmente ou sincronizadas com a conta na *cloud*, assim é permitido o acesso a partir de diferentes dispositivos. É possível definir testes automáticos através de *scripts* JavaScript que validam códigos de resposta, estrutura de dados e valores específicos. A ferramenta suporta execução em lote (*collection runner*) e exportação dos resultados. Apesar de não ser *open-source*, a versão gratuita revelou-se adequada para este contexto.

No *frontend* foram utilizadas as ferramentas Jasmine e Karma, integradas por omissão no Angular CLI. O Jasmine fornece métodos como `describe(), it() e expect()` para estruturar e executar os testes, enquanto o Karma gere a sua execução em diferentes *browsers*. A *framework* suporta o uso de `TestBed` para configurar módulos de teste e criar *mocks* de serviços. A execução dos testes é realizada através do comando `ng test`, que inicia um servidor de desenvolvimento e corre os testes em modo *watch*. Esta configuração permite validar desde métodos isolados até fluxos completos de API, cobrindo diferentes camadas da aplicação.

4.4. Metodologia, Levantamento e Redesenho do Processo de Requisição de Viatura

Esta secção descreve a metodologia adotada no levantamento e análise do processo atual de autorização de condução de viatura da ESTG, bem como o redesenho proposto com base nos princípios do ciclo de vida BPM. O conteúdo inclui o enquadramento metodológico, a caracterização do fluxo operacional, a identificação dos intervenientes, as limitações observadas e as melhorias introduzidas na versão redesenhada do processo.

4.4.1. Metodologia e Enquadramento no Ciclo de Vida BPM

O levantamento e análise do processo de autorização de condução de viatura da ESTG foram conduzidos de acordo com uma abordagem metodológica estruturada, alinhada com os princípios e fases do ciclo de vida BPM apresentados no Capítulo 2. Este enquadramento teórico permitiu a coerência entre o modelo conceptual descrito e a sua aplicação prática no contexto institucional, assegurando a conformidade metodológica com as boas práticas de modelação, análise e melhoria de processos.

A metodologia seguida foi concebida para apoiar as etapas de identificação, descoberta, análise, redesenho, implementação e monitorização, que compõem o ciclo BPM. A Figura 8 apresenta o modelo metodológico adotado, representando as fases de desenvolvimento aplicadas à análise e transformação do processo de autorização de condução de viatura.

Figura 8 - Enquadramento metodológico no ciclo de vida BPM – requisição de viatura

O diagrama apresentado integra as diferentes fases do ciclo, destacando a progressão lógica entre as etapas de identificação e descoberta, que marcam o início do processo analítico, e as de monitorização e melhoria contínua que refletem o carácter cíclico da BPM. As setas bidirecionais entre as etapas evidenciam o carácter iterativo e dinâmico, indicam que as fases não ocorrem de forma linear, mas sim com retornos quando necessário. Esta natureza recursiva permite ajustar continuamente o processo de acordo com os resultados de desempenho obtidos o que possibilita a aprendizagem organizacional e a evolução progressiva do modelo de gestão.

A aplicação deste enquadramento metodológico fundamentou o planeamento das atividades descritas nas secções seguintes, orientando a execução do levantamento empírico do processo e a sua posterior modelação “as-is”. Assim, a abordagem utilizada permitiu não apenas a coerência teórica com o ciclo BPM, mas também a estruturação prática necessária para o desenvolvimento de um processo de análise e redesenho sustentado.

4.4.2. Levantamento do Processo

O levantamento foi conduzido através de uma abordagem metodológica que combinou três técnicas complementares de recolha de dados:

- Análise documental do procedimento ESTG-PS03V22, formulários de autorização e ferramentas de registo utilizadas.
- Entrevistas exploratórias com colaboradores do Secretariado da Presidência responsáveis pela gestão operacional do processo.
- Observação direta do funcionamento quotidiano, incluindo interações entre intervenientes e utilização de documentação.

Esta metodologia permitiu obter uma visão abrangente e validada do processo atual, identificando tanto os procedimentos formais como as práticas operacionais efetivamente implementadas.

4.4.3. Caracterização do Processo Atual

O processo de autorização de condução de viatura encontra-se implementado de forma integralmente manual, opera através de documentação física e comunicação telefónica entre os intervenientes. A Figura 9 mostra o fluxo atual do processo, evidenciando as quatro etapas principais definidas no procedimento ESTG-PS03V22.

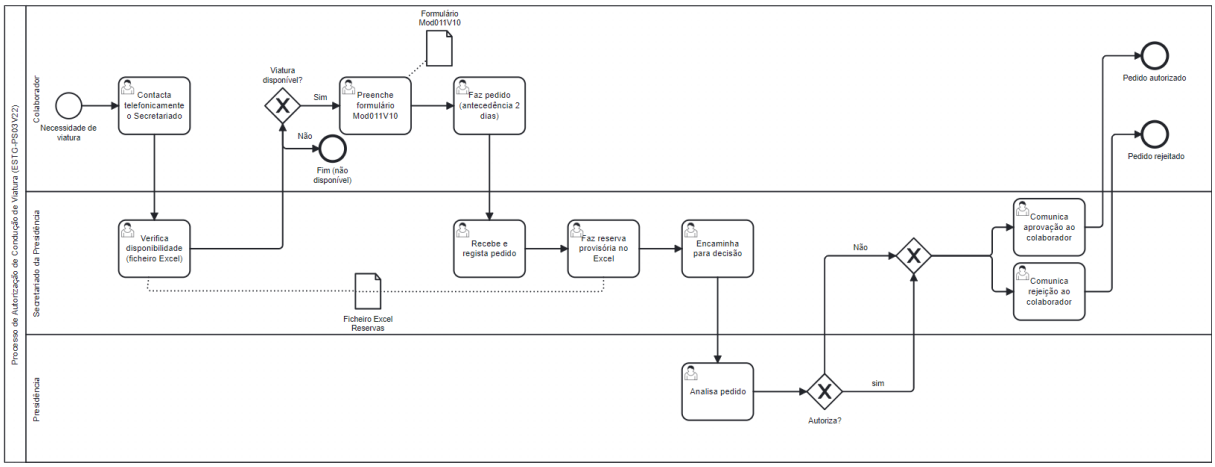


Figura 9 - Diagrama BPMN requisição de viatura – processo atual

O processo inicia-se com o pedido de autorização pelo colaborador, que contacta telefonicamente o Secretariado da Presidência para verificar disponibilidade e posteriormente entrega o formulário preenchido. Este pedido contempla três modalidades distintas: condução do veículo de serviços gerais da ESTG, utilização de automóvel próprio nos termos dos números 2 e 3 do artigo 20º do Decreto-Lei nº 106/98, ou utilização de automóvel próprio nos termos do número 4 do mesmo diploma.

Segue-se a verificação e reserva, onde o Secretariado consulta manualmente o ficheiro Excel de controlo de disponibilidade e procede ao registo da nova reserva. O despacho de autorização é da competência da Presidência, que decide pela autorização ou não autorização do pedido. Finalmente, o Secretariado comunica a decisão ao colaborador.

4.4.4. Atores e Responsabilidades

O processo envolve três categorias de intervenientes com responsabilidades claramente definidas. O colaborador requerente é responsável pela submissão atempada do pedido (antecedência mínima de 2 dias úteis), preenchimento do formulário com dados da deslocação e, no caso de utilização de viatura institucional, pelo registo de quilometragem e eventuais ocorrências no final da utilização. O Secretariado da Presidência atua como entidade coordenadora, sendo responsável pela verificação de disponibilidade de viaturas, registo de reservas, encaminhamento de pedidos para decisão e comunicação dos resultados aos requerentes. A Presidência detém a competência decisão final sobre a autorização dos pedidos, considerando critérios como a necessidade de serviço, disponibilidade de recursos e conformidade com as normas institucionais.

4.4.5. Limitações Identificadas

A análise revelou limitações significativas que comprometem a eficiência operacional e justificam a necessidade de digitalização do processo. A dependência de comunicação manual constitui um ponto crítico, uma vez que a verificação inicial de disponibilidade baseia-se em contato telefónico, o que cria dependência da disponibilidade imediata dos colaboradores e potencial para falhas de comunicação. A gestão manual de reservas através de ficheiro Excel apresenta riscos inerentes de inconsistência, sobreposições e perda de dados, não permitindo consulta em tempo real por múltiplos utilizadores. A ausência de automatização manifesta-se na inexistência de notificações automáticas, *dashboards* de monitorização ou mecanismos sistemáticos de auditoria, enquanto as limitações de escalabilidade evidenciam que o modelo atual aloca uma pessoa para gerir todas esta atividade.

Esta análise do processo atual estabelece a base para a identificação dos requisitos funcionais e não funcionais que orientarão o desenvolvimento da solução proposta.

4.4.6. Redesenho do Processo de Requisição de Viatura e o seu impacto Operacional

Com base na análise do processo atual apresentada na secção anterior, o processo de requisição de viatura foi redesenhado. O novo modelo elimina as limitações identificadas e introduz automação através de tecnologias de *workflow*. A Figura 10 apresenta o modelo BPMN do processo redesenhado.

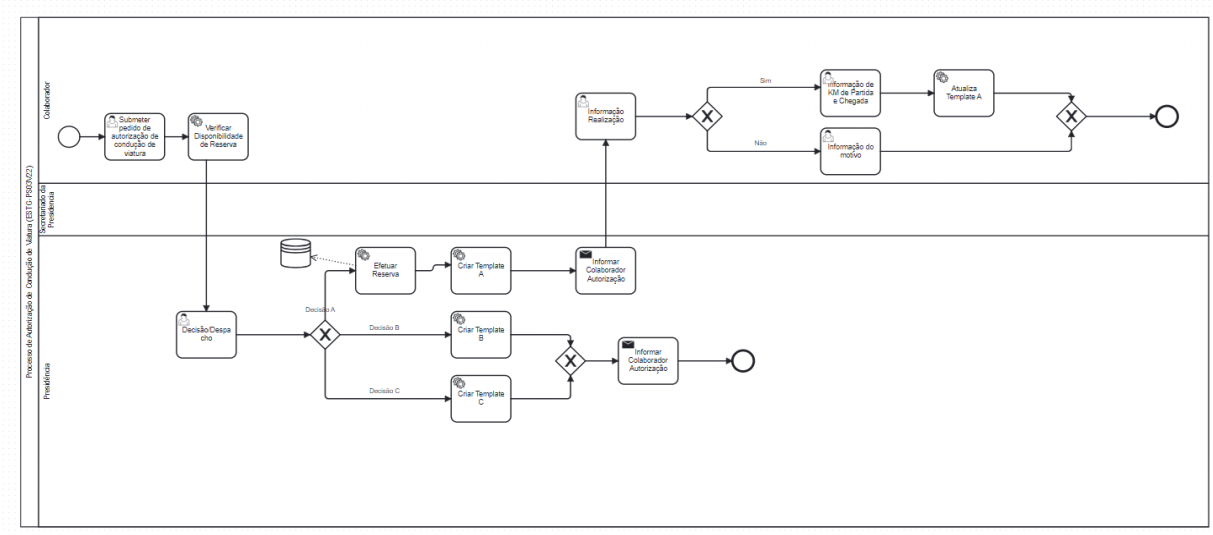


Figura 10 - Diagrama BPMN requisição de viatura – processo redesenhado

O processo redesenhado introduz submissão digital direta. O colaborador submete o pedido em formulário estruturado com validação automática, eliminando a dependência telefónica do secretariado e permite acesso fora do horário de funcionamento. Esta alteração reduz significativamente a carga de trabalho manual e melhora a experiência do utilizador.

A verificação de disponibilidade torna-se automática. Esta alteração elimina consultas manuais ao Excel, reduz erros de sobreposição e fornece resposta imediata sobre conflitos ou disponibilidade. O *feedback* imediato acelera o tempo de resposta e permite ajustes antes da submissão final.

O encaminhamento do processo utiliza *gateways* exclusivos que direcionam automaticamente conforme o tipo de autorização. Cada modalidade segue o *template* apropriado, garantindo conformidade com o procedimento institucional. A geração de documentos passa de impressão física

para formato digital. O sistema produz documentos PDF formatados e envia notificações por email, eliminando impressão, arquivo manual e comunicação telefónica.

Esta digitalização elimina todo o trabalho manual realizado pelo secretariado, proporciona visibilidade completa do estado dos pedidos, facilitando gestão e acompanhamento. Esta transparência suporta identificação de gargalos e otimização contínua.

4.5. Arquitetura Proposta

A implementação do processo redesenhado em BPMN requer uma arquitetura técnica que suporte a automação e integração identificadas. A solução proposta baseia-se numa arquitetura de microserviços que materializa os princípios e padrões apresentados nas secções anteriores. A definição da arquitetura segue uma abordagem estruturada que parte da identificação de contextos de domínio, procede ao mapeamento para microserviços específicos e culmina numa visão integrada que demonstra como os componentes interagem para suportar o processo automatizado.

4.5.1. Bounded Contexts e Decomposição

A definição da arquitetura da solução baseou-se na aplicação dos princípios de Domain-Driven Design, conforme abordado na secção 4.1.1, para identificar os *bounded contexts* que melhor representam o domínio do processo de autorização de condução de viatura. A análise do domínio resultou na identificação de cinco bounded contexts distintos, cada um encapsula responsabilidades específicas e mantem autonomia sobre o seu modelo de dados e regras de negócio. Apresentam-se de seguida os contextos identificados e as suas características principais:

- ***Context de Gestão de Identidade:*** Este contexto engloba toda a lógica relacionada com autenticação, autorização e gestão de utilizadores. É responsável pela validação de credenciais através de integração LDAP com o Active Directory institucional, emissão e validação de *tokens* JWT, e controlo de acesso baseado em funções. A separação deste contexto garante que as preocupações de segurança permanecem isoladas e centralizadas, o que facilita auditorias e cumprimento de políticas de segurança.
- ***Context de Processos de Autorização de Condução de Viatura:*** Constitui o núcleo do domínio da aplicação, concentra toda a lógica de negócio relacionada com o ciclo de vida dos pedidos de autorização. Inclui a submissão de pedidos, encaminhamento para decisão, registo de despachos e acompanhamento de realizações. Este contexto mantém o conhecimento

específico do processo e das regras de negócio institucionais, constituindo o *bounded context* principal da solução.

- ***Context de Gestão de Recursos*:** Dedica-se exclusivamente à gestão das viaturas institucionais, incluindo registo de veículos, controlo de disponibilidade e gestão de reservas. A separação deste contexto permite evolução independente das funcionalidades relacionadas com recursos, facilitando futuras extensões como gestão de combustível, manutenções ou integração com sistemas de *tracking*.
- ***Context de Comunicação e Notificações*:** Centraliza todas as funcionalidades relacionadas com comunicação externa, incluindo envio de notificações por email. A autonomia deste contexto permite implementar diferentes canais de comunicação sem impacto nos restantes componentes da solução.
- ***Context de Documentação*:** Responsável pela geração de documentos oficiais em formato PDF, seguindo *templates* institucionais estabelecidos. A separação permite evolução independente das capacidades de geração documental e facilita integração com diferentes sistemas de gestão documental.

4.5.2. Mapeamento para Microserviços

A decomposição dos *bounded contexts* resulta na estrutura de microserviços apresentada na Tabela 3.

BOUNDED CONTEXT	MICROSERVIÇO(S)	RESPONSABILIDADE PRINCIPAL
GESTÃO DE IDENTIDADE	Identity Server Users API	Autenticação/autorização Gestão de dados de utilizadores
AUTORIZAÇÃO DE CONDUÇÃO DE VIATURA	Demo API	Orquestração do processo de negócio
GESTÃO DE RECURSOS	Vehicles API	Gestão de viaturas e disponibilidade
COMUNICAÇÃO E NOTIFICAÇÕES	NotifyFlow API	Envio de notificações e alertas
DOCUMENTAÇÃO	PDF Generator Service	Geração de documentos oficiais

Tabela 3 - Mapeamento de bounded contexts para microserviços

Esta decomposição permite que cada microserviço possua responsabilidades definidas, o que facilita o desenvolvimento, teste e manutenção independentes. A comunicação entre *bounded contexts* é

realizada através de interfaces convenientemente definidas, preservando a autonomia de cada contexto enquanto assegura a integração com a solução global.

4.5.3. Visão Geral da Arquitetura

A arquitetura da solução proposta materializa os *bounded contexts* identificados numa estrutura distribuída que integra os padrões com as tecnologias selecionadas conforme descrito na secção 4.3.

A Figura 11 apresenta a organização geral dos componentes e as suas interações principais, estruturada em camadas que refletem diferentes níveis de abstração e responsabilidade na solução.

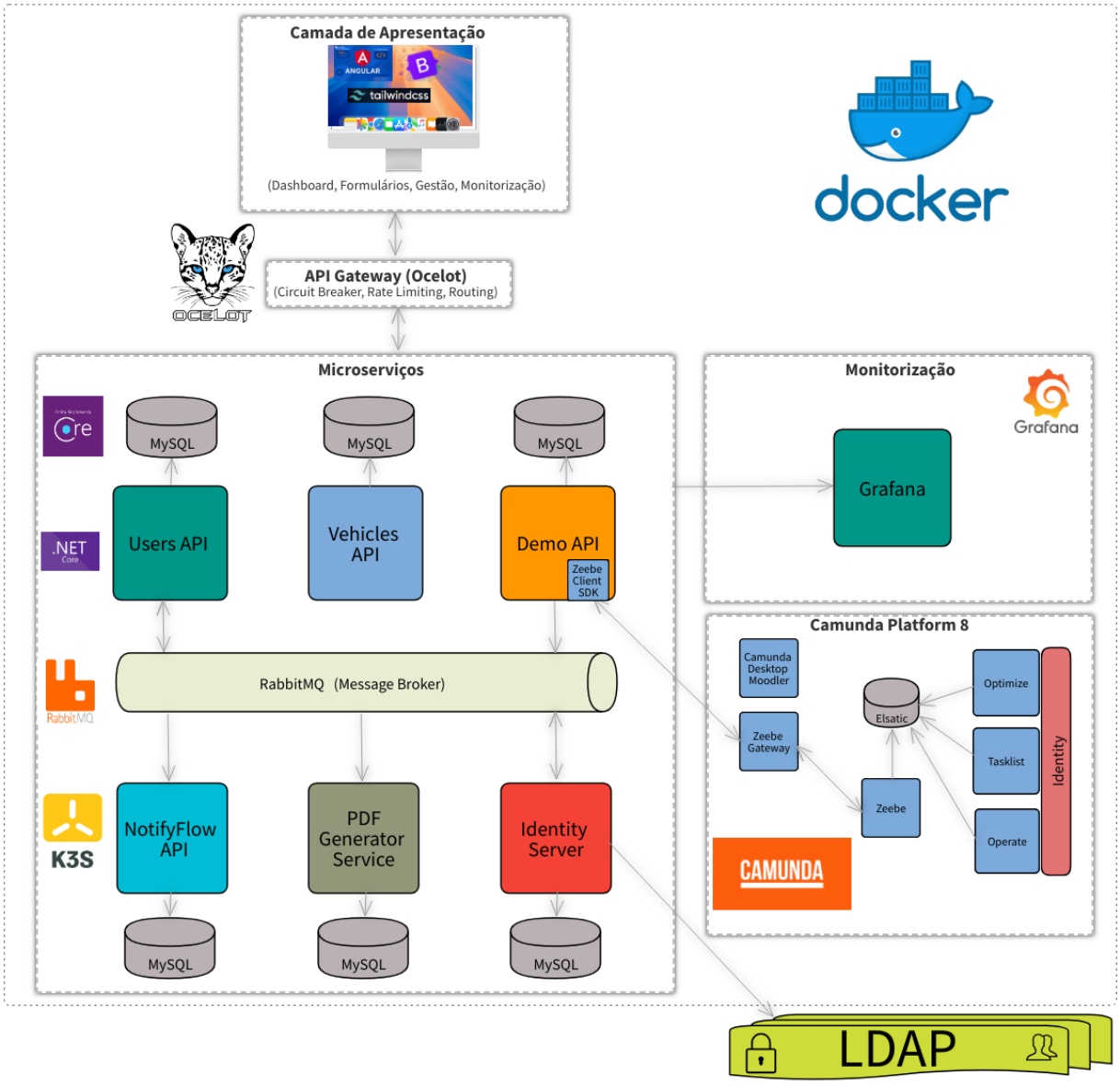


Figura 11 - Arquitetura geral da solução

A camada de apresentação disponibiliza uma aplicação Angular responsiva que adapta a interface conforme o perfil do utilizador autenticado. Colaboradores acedem a funcionalidades de submissão de pedidos, consulta de estado e registo de realizações, enquanto decisores dispõem de *dashboards* de gestão de aprovações. A aplicação comunica através do API Gateway, mantendo o princípio de ponto único de acesso estabelecido.

O API Gateway, materializa os padrões de Circuit Breaker e centralização de acesso. Este componente interceta todos os pedidos externos, aplica políticas de *rate limiting*, e encaminhamento dinâmico para os microserviços apropriados. A camada de microserviços operacionaliza os *bounded contexts* identificados através de seis serviços autónomos. Interações síncronas através de API REST são utilizadas para operações que requerem resposta imediata, como validação de *tokens*, consultas de disponibilidade e operações de leitura em geral. A comunicação assíncrona é feita através do RabbitMQ que suporta operações onde a resposta imediata não é crítica, incluindo eventos de criação e sincronização de utilizadores entre *Identity Server* e *Users API*, envio de notificações e geração de documentos. Esta abordagem mantém a resiliência do sistema face a indisponibilidades temporárias.

A camada de dados implementa o padrão Database per Service, com cada microserviço mantém autonomia sobre o seu modelo de dados. As instâncias MySQL são isoladas por serviço, o que elimina dependências de dados.

A integração com a plataforma Camunda 8 é realizada através do Demo API, que incorpora o *Zeebe* Client SDK para comunicação com o motor de workflow. O Demo API utiliza o *ZeebeService* para coordenar a execução dos processos BPMN. Os sistemas de monitorização, compostos pelo *Grafana*, proporcionam visualização de *dashboards* e análise de dados estruturados.

A autenticação integra-se com o Active Directory através de LDAP no Identity Server. Quando um utilizador tenta aceder ao sistema, o Identity Server consulta primeiro a base de dados local e, se não encontrar o utilizador, valida as credenciais no servidor LDAP configurado.

A arquitetura combina autonomia de serviços com coordenação do processo de negócio. Esta estrutura suporta os requisitos atuais e permite evolução e adaptações futuras.

5. Implementação

Este capítulo descreve a implementação da arquitetura proposta no capítulo anterior. São apresentados os microserviços desenvolvidos, a implementação dos *patterns* de integração, a containerização e orquestração, o desenvolvimento do *frontend* e os resultados da monitorização.

O capítulo inicia com os microserviços implementados e sua comunicação, seguindo-se a implementação dos *patterns* de integração, as migrações para *Docker* e K3s, o desenvolvimento da camada de apresentação, a análise de dados através do *Grafana* e finaliza com os testes e documentação.

5.1. Microserviços Implementados e sua comunicação

Esta secção apresenta a implementação dos microserviços que compõem a solução, detalhando as suas responsabilidades específicas, padrões de comunicação e integrações realizadas. Os diagramas de sequência apresentados focam-se na lógica interna de cada microserviço, omitindo o *API Gateway* para simplificação, embora todas as comunicações externas sejam mediadas por esta camada conforme ilustrado na arquitetura geral.

5.1.1. Identity Server

O Identity Server centraliza a gestão de identidade e autorização da solução, implementado com base no *Duende IdentityServer* integrado com *ASP.NET Core Identity*. A configuração suporta os protocolos *OAuth 2.0* e *OpenID Connect* através de múltiplos clientes configurados para *Resource Owner Password Credential Grant* e *Authorization Code Flow*. O sistema incorpora auditoria através da ativação de eventos (RaiseErrorEvents, RaiseInformationEvents, RaiseFailureEvents, RaiseSuccessEvents), o que permite rastreabilidade das operações de autenticação.

5.1.1.1. Autenticação Híbrida

A implementação da classe `HybridResourceOwnerPasswordValidator` segue uma estratégia de validação em duas fases. Primeiro, executa verificação local através dos métodos `FindByNameAsync()` e `CheckPasswordAsync()` do *ASP.NET Core Identity*. Em caso de insucesso,

o sistema executa *fallback* para o *Active Directory* institucional através de protocolo *Lightweight Directory Access Protocol* (LDAP).

A integração LDAP implementa múltiplos formatos de *binding* administrativo e filtros de pesquisa (sAMAccountName, userPrincipalName, uid) para compatibilidade com diferentes topologias de diretório. O método `TryLdapLogin()` executa *binding* inicial com credenciais administrativas, seguido de pesquisa do utilizador e validação direta das suas credenciais através de novo *binding* com o *distinguished name* obtido.

A proteção de credenciais administrativas utiliza a classe `LdapPasswordProtector`, que implementa encriptação através da biblioteca `Microsoft.AspNetCore.DataProtection`. Este mecanismo inclui o método `ReencryptAllLdapPasswordsAsync()` para rotação automática de chaves de segurança, garantindo que *passwords* LDAP armazenadas na base de dados permaneçam protegidas mesmo em cenários de comprometimento.

5.1.1.2. Implementação do *Access Token Pattern*

A estrutura de autorização baseia-se em *scopes* granulares que permitem autorização distribuída onde cada microserviço valida autonomamente os *tokens* através de verificação criptográfica da assinatura JWT. A configuração estabelece *scopes* específicos por microserviço (users.api, vehicles.api, notifyflow.api, demo.api, pdf.api) e operações transversais (read, write, delete), conforme apresentado na Figura 12.

```
01 public static IEnumerable<ApiScope> ApiScopes =>
02     new List<ApiScope>
03     {
04         new ApiScope("users.api", "Users API"),
05         new ApiScope("vehicles.api", "Vehicles API"),
06         new ApiScope("notifyflow.api", "NotifyFlow API"),
07         new ApiScope("demo.api", "Demo API"),
08         new ApiScope("pdf.api", "PDF Generator API"),
09         new ApiScope(name: "read", "Read data."),
10         new ApiScope(name: "write", "Write data."),
11         new ApiScope(name: "delete", "Delete data.")
               12     };
```

Figura 12 - Configuração de scopes para access tokens

Os microserviços validam *tokens* através do *middleware* JWT que verifica a assinatura digital contra a chave pública do IdentityServer. A autorização implementa-se através de políticas que verificam *scopes* específicos no *token*, conforme exemplificado na Figura 13.

```
01 builder.Services.AddAuthentication("Bearer")
02     .AddJwtBearer("Bearer", options =>
03     {
04         options.Authority = builder.Configuration["Proj.IdentityServer:ApplicationUrl"];
```

```
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateAudience = false  // Desenvolvimento
        };
    });

builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("UsersApiPolicy", policy =>
        policy.RequireAuthenticatedUser()
              .RequireClaim("scope", "users.api"));
});
```

Figura 13 - Validação JWT e políticas de autorização e autenticação

Esta implementação permite que cada microserviço valide *tokens* de forma autónoma através da verificação criptográfica, sem necessitar de comunicação com o servidor de identidade a cada pedido. Para produção, a configuração `ValidateAudience` será ativada para verificar que o *token* foi emitido especificamente para o serviço que o está a validar, prevenindo ataques onde *tokens* válidos são usados em serviços não autorizados. O tempo de vida dos *access tokens* é reduzido para 10-30 minutos.

A implementação do *Access Token Pattern* estabeleceu assim uma camada de segurança distribuída e escalável, onde cada microserviço mantém autonomia na validação e autorização, sem comprometer a segurança ou criar dependências síncronas entre serviços.

5.1.1.3. Comunicação e Fluxo de Processo

A integração de utilizadores LDAP implementa padrão assíncrono que cria contas locais para utilizadores autenticados externamente, mas inexistentes na base de dados interna. O processo utiliza comunicação baseada em eventos através de *message broker*, que desacopla a validação LDAP da criação local de utilizadores. O sistema prioriza disponibilidade sobre consistência transacional, abordagem que permite recuperação de falhas através de tentativas subsequentes de autenticação. Esta estratégia adequa-se a ambientes onde a continuidade operacional supera a necessidade de garantias transacionais.

A autenticação segue especificação *OAuth* 2.0 através de *endpoint* dedicado que processa credenciais e emite *tokens* JWT padronizados. Este mecanismo suporta tanto utilizadores locais como federados, proporciona uma interface unificada independentemente da origem dos dados de utilizador. O fluxo de exemplo deste processo de autenticação é detalhado na Figura 14.

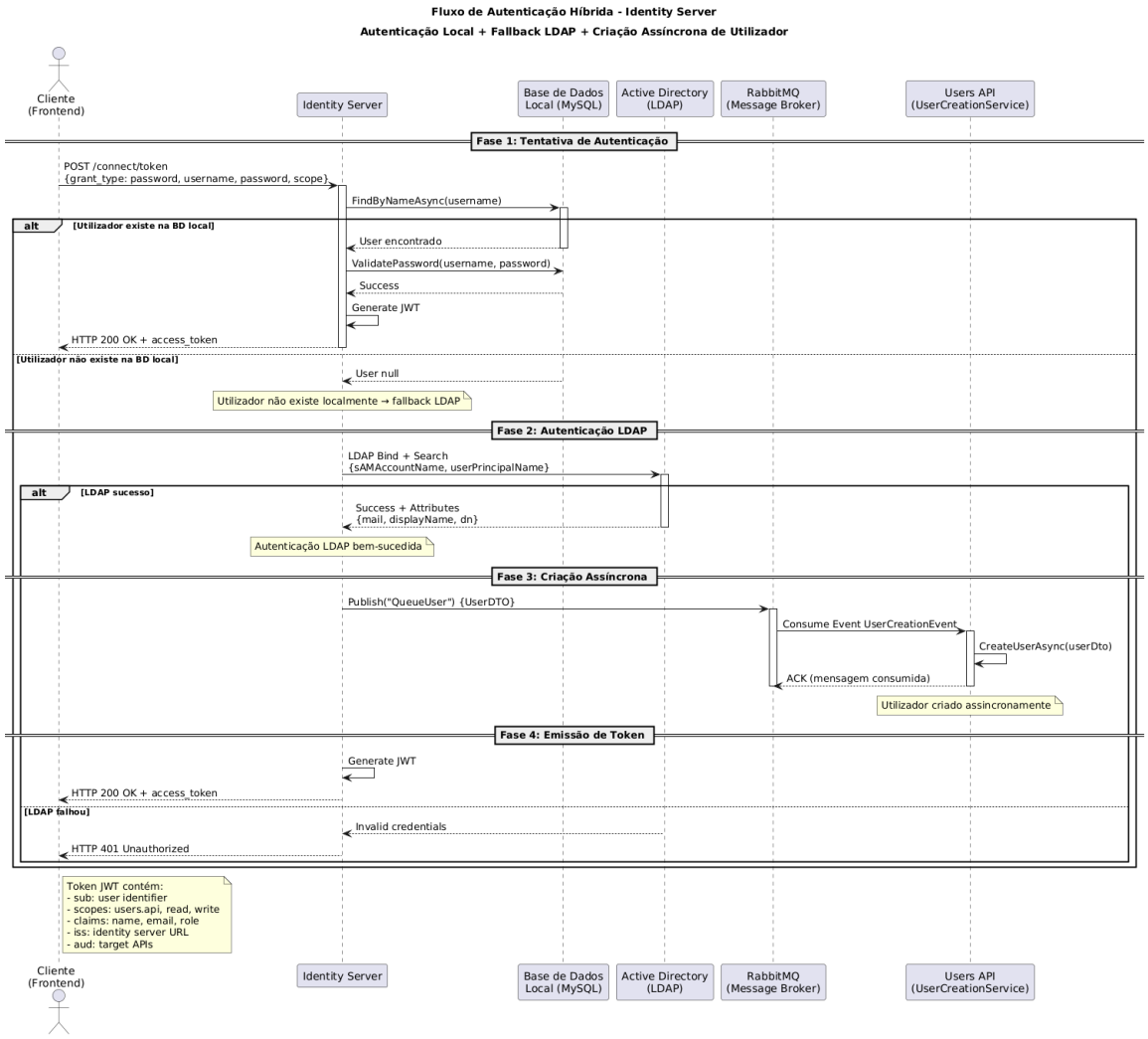


Figura 14 - Diagrama de sequência da autenticação híbrida do Identity Server

O fluxo demonstra a coordenação entre componentes durante quatro fases distintas: verificação local na base de dados *MySQL* através do *Entity Framework* , *fallback* LDAP com *binding* direto utilizando `System.DirectoryServices.Protocols`, criação assíncrona via RabbitMQ *message broker*, e emissão de *tokens* JWT com *scopes* e *claims* configurados. Esta arquitetura permite autenticação transparente mantendo separação de responsabilidades entre microserviços e tolerância a falhas através de comunicação assíncrona.

5.1.2. Users API

O Users API constitui o microserviço responsável pela gestão de utilizadores, grupos e perfis na solução. Implementado com *ASP.NET Core*, este serviço centraliza todas as operações relacionadas com a administração de identidades, permite operações de CRUD através de uma arquitetura baseada

em repositórios e *Data Transfer Objects* (DTO). O microserviço implementa o padrão *Database per Service*, mantendo autonomia sobre o seu modelo de dados através de uma base de dados *MySQL* dedicada. Os *endpoints* principais do Users API encontram-se no Apêndice A, incluindo as operações de criação, consulta, atualização e eliminação de utilizadores.

5.1.2.1. Responsabilidades e Funcionalidades

O microserviço expõe três *controllers* `UsersController`, `GroupsController` e `RolesController`, Cada *controller* oferece operações de CRUD para as respetivas entidades, incluindo funcionalidades como pesquisa por critérios específicos e gestão de associações entre utilizadores e grupos.

A validação de integridade é implementada através do método `ValidatePassword()`, que aplica políticas de segurança configuráveis. Por omissão, o sistema exige um mínimo de 12 caracteres, incluindo obrigatoriamente letras maiúsculas e minúsculas, dígitos numéricos e caracteres especiais, sendo estes critérios parametrizáveis conforme as políticas de segurança definidas pela organização. Adicionalmente, o sistema valida a unicidade de e-mail e *username* através de verificações na base de dados antes da criação de novas contas, prevenindo duplicações e garantindo a integridade referencial.

A gestão de grupos e roles permite organização hierárquica de utilizadores, com funcionalidades para associar utilizadores a múltiplos grupos e definir permissões através de *roles* específicas. O método `GetGroupNamesByUserId()` permite consultar todas as associações de um utilizador, suportando sistemas de autorização baseados em grupos. A arquitetura de repositórios (IUserRepository, IGroupRepository, IRoleRepository) implementa o padrão *Repository Pattern* que encapsula a lógica de acesso a dados e proporciona uma abstração sobre a *Entity Framework Core*. Esta abordagem permite a separação entre a lógica de negócio nos serviços e as operações de persistência.

A evolução do esquema da base de dados é gerida através do sistema de *migrations* do *Entity Framework Core*, o que permite controlo de versões das alterações. As *migrations* são aplicadas durante o arranque da aplicação, o que assegura que a estrutura da base de dados está sincronizada com o modelo de entidades definido no código.

5.1.2.2. Comunicação e Fluxo de Processo

A comunicação com o Identity Server é realizada através de RabbitMQ utiliza a fila `QueueIdentity`. Quando um utilizador é criado o sistema executa validações de integridade, persiste os dados na base de dados *MySQL* local, e publica um evento `UserDTO` serializado para sincronização com o *Identity Server*. O método `SendMessageToRabbitMQ()` utiliza `BasicPublish()` para permitir entrega da mensagem, enquanto o *AutoMapper* assegura conversão entre entidades de domínio e DTO. O fluxo deste processo de criação de utilizadores é detalhado na Figura 15.

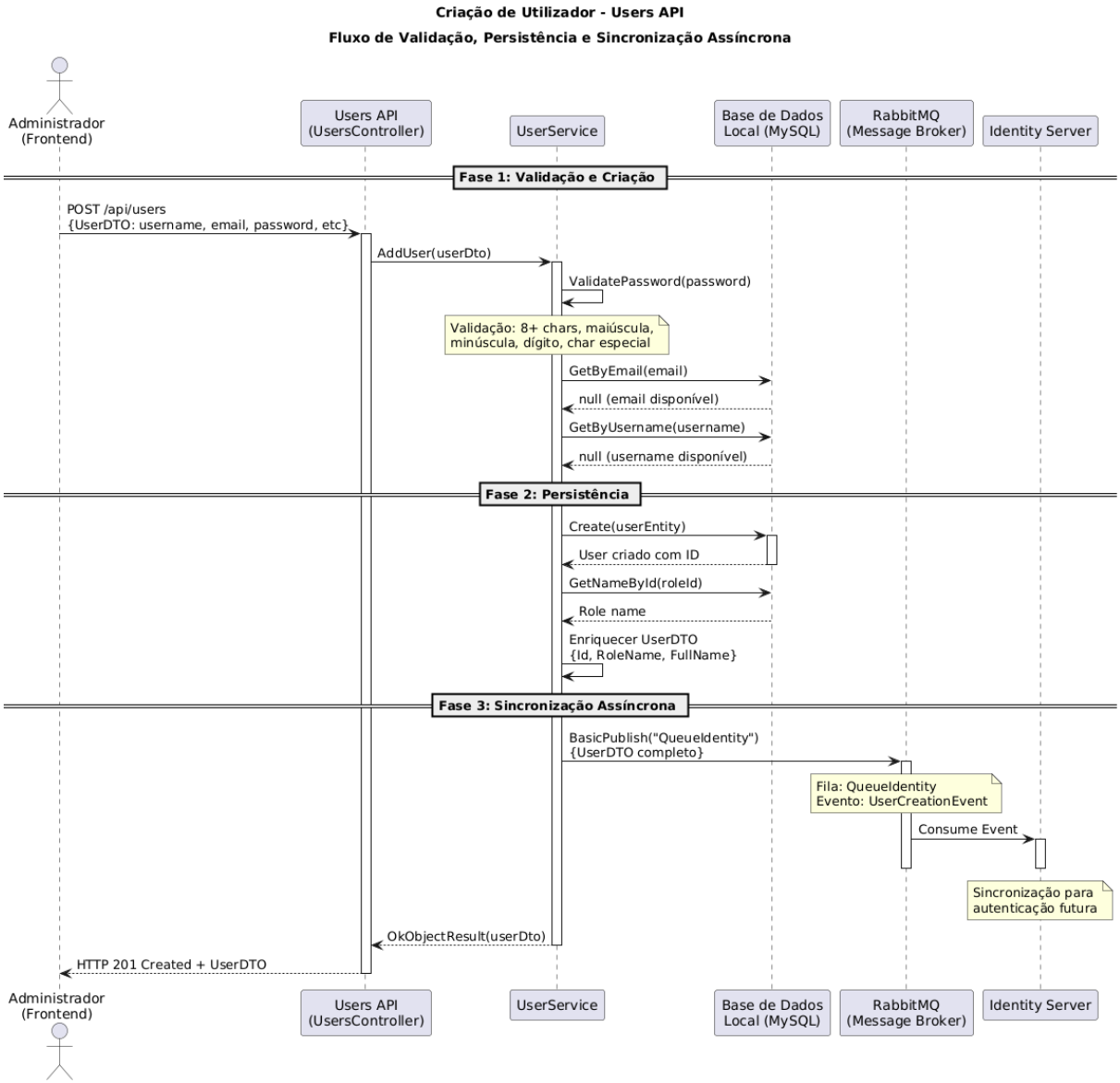


Figura 15 - Diagrama de sequência criação de utilizador

O fluxo demonstra um exemplo do processo de criação de utilizador por parte de um administrador, evidenciando como o microserviço coordena validações locais, persistência de dados e comunicação assíncrona. A arquitetura assegura consistência de dados mantendo autonomia do microserviço e propagação dos eventos para sincronização distribuída.

5.1.3. Demo API

O Demo API representa o microserviço central da solução, responsável pela orquestração dos processos de negócio através da integração com a plataforma Camunda 8. Este microserviço materializa o *bounded context* de Autorização de Condução de Viatura, concentrando a lógica do domínio e coordenando as interações com os restantes serviços. A implementação baseia-se em ASP.NET Core e utiliza o *Zeebe* Client para comunicação com o motor de *workflow*. A especificação da interface REST encontra-se detalhada no Apêndice B.

5.1.3.1. Integração com Camunda 8 e Configuração do Cliente

A integração com o Camunda 8 é configurada através do registo do `IZeebeClient` como *Singleton* no container de dependências, isto é, cria apenas uma instância do cliente *Zeebe* durante todo o ciclo de vida da aplicação e essa instância é reutilizada em vez de criar várias. O cliente utiliza `ZeebeClient.Builder()` com configuração para desenvolvimento, incluindo `UseLoggerFactory(new NLogLoggerFactory())` para integração com o sistema de *logging*. A *gateway address* é obtida da configuração `ZEEBE_ADDRESS` configurado no `appsettings.json`, o que permite flexibilidade entre ambientes.

O `ZeebeService` recebe este cliente através de injeção de dependências e coordena tanto o *deployment* inicial como a execução de *workflows*. Durante a inicialização, o serviço regista automaticamente os *workers* através do `ZeebeWorkerManager` para *Jobs* e executa *deployment* do processo BPMN através de `Task.Run(),GetAwaiter(),GetResult()` síncrono no *startup*, para permitir que o processo está disponível antes da aplicação aceitar pedidos.

O método `Deploy()` utiliza `NewDeployCommand()` para carregar ficheiros BPMN do diretório *Resources*, e armazena o `processDefinitionKey` retornado para criação posterior de instâncias. O `ZeebeWorkerManager` implementa um dicionário de `handlers (_jobHandlers)` que mapeia os nomes de métodos para os *delegates* `JobHandler`, o que permite associação dinâmica entre `jobTypes` e lógica de processamento através do método `StartWorker()`.

5.1.3.2. Controlo de User Tasks e Mecanismo de Processamento

O `ZeebeController` expõe *endpoints REST* para coordenação das *user tasks* do processo *BPMN*. O método `CompleteTask()` implementa um mecanismo híbrido que combina criação de instâncias com processamento de *user tasks*. Para novas submissões (`taskId == 0`), o método invoca `StartWorkflowInstance()` e cria um *worker* temporário do tipo `"io.camunda.Zeebe:userTask”`. Para tarefas existentes, utiliza o `processInstanceKey` para localizar e completar a tarefa que já foram iniciadas. O método `HandleJob()` processa *jobs* baseado no `ElementId`, que implementa a lógica condicional para diferentes pontos do processo, nomeadamente para submissões iniciais, para aprovações, registo de realizações, para justificações de não realização, para quilometragem. Cada tipo de tarefa executa `NewSetVariablesCommand()` que permite atualizar as variáveis do processo seguido de `CompleteJobCommand()` para avançar o *workflow*.

Os *endpoints* `userTaskSubmit`, `userTaskDecision` e `UserTaskRealization` recebem objetos da reserva, executam validações específicas de controlo de dados e sincronizam dados com a base de dados local através do `ReservationService`.

5.1.3.3. Workers e Comunicação Assíncrona

O `ZeebeWorkerManager` coordena *external task workers* que executam através de arquitetura assíncrona com controlo de concorrência configurável. A parametrização dos *workers* estabelece limites de processamento paralelo, intervalos de verificação periódica, e tempos de expiração que optimizam *throughput* e resiliência operacional. Os *workers* implementam conclusão automática após execução, que simplifica gestão de ciclo de vida das tarefas.

A comunicação assíncrona utiliza padrão baseado em filas através de RabbitMQ. Os *handlers* para *templates* (A, B, C) coordenam geração de documentos diferenciados por tipo de autorização através de fila dedicada, enquanto o *handler* de notificações processa comunicações externas e o *handler* de atualização gere modificações documentais pós-execução.

Esta arquitetura híbrida coordena execução síncrona de *workflow Zeebe* com comunicação assíncrona distribuída, que mantém desacoplamento temporal e autonomia dos microserviços. A Figura 16 demonstra o fluxo completo de um processo de requisição de viatura, evidenciando a coordenação entre componentes durante as diferentes fases do *workflow*.

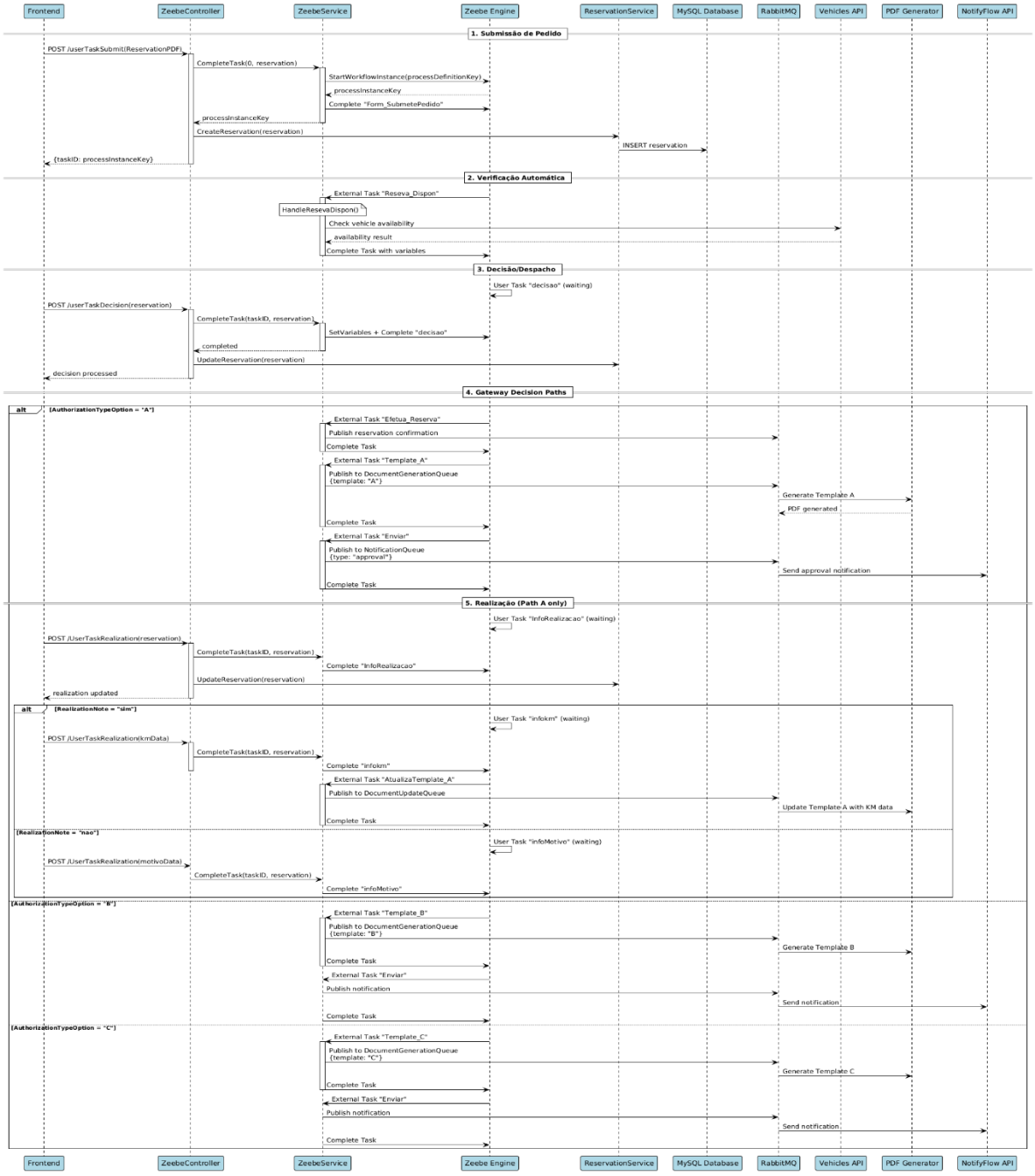


Figura 16 - Diagrama de sequência processo de requisição de viatura

O fluxo mostra a coordenação entre componentes durante as cinco fases principais do processo: submissão de pedido com criação de instância *Zeebe*, verificação automática de disponibilidade através do *Vehicles API*, decisão/despacho com *user task* para aprovação, processamento diferenciado por *paths* (A, B, C) com geração de *templates* e notificações via RabbitMQ, e gestão da fase de realização exclusiva do *path* A com possível atualização de documentos baseada na confirmação de execução. A arquitetura assegura consistência entre o estado do *workflow* e os dados da aplicação

através de sincronização entre variáveis *Zeebe* e persistência local, mantendo autonomia dos microserviços especializados enquanto garante progressão coordenada do processo de negócio.

5.1.4. PDF Generator Service

O PDF Generator Service constitui um microserviço responsável pela geração de documentos PDF formatados, responsável por materializar os *templates* de autorização definidos no processo BPMN. Implementado como uma aplicação ASP.NET Core, este serviço processa requisições assíncronas recebidas através de RabbitMQ e produz documentos oficiais baseados em *templates* HTML pré-configurados.

A arquitetura do serviço baseia-se na biblioteca iText7 para conversão HTML-PDF, utiliza a classe PdfService que encapsula as operações de carregamento de *templates*, substituição de *placeholders* e geração de documentos. O *template* HTML (`request-template.html`) define a estrutura visual do documento, incluindo logótipo institucional, formatação de cabeçalho com informações de decisão/despacho, e secções para dados da requisição e informações de realização. O Apêndice C apresenta um exemplo de um PDF gerado. O `PdfController` expõe dois *endpoints* principais, `Generatepdf` que aceita objetos `RequestCar` com dados da reserva e persiste o PDF gerado com identificador único. O *endpoint* `GetPdf` permite recuperação de documentos previamente gerados, retornando o ficheiro PDF como resposta binária com *content-type* apropriado.

Os documentos PDF gerados são persistidos como ficheiros no sistema de ficheiros (diretório `GeneratedPdfs`) em vez de armazenamento na base de dados, uma decisão que privilegia performance de acesso, redução de *overhead* de *queries* SQL. A base de dados MySQL dedicada está preparada para armazenar definições de *templates* (conteúdo HTML, *placeholders* disponíveis, tipos de autorização), metadados de *placeholders* (nome, tipo de dados, valores por defeito) e histórico de documentos gerados para auditoria, permitindo futura implementação de editor visual no *frontend* para criação e personalização dinâmica de *templates*.

5.1.5. Vehicles API

O Vehicles API constitui um microserviço dedicado à gestão de viaturas institucionais, implementando as operações *CRUD* necessárias ao controlo do parque automóvel. Este microserviço representa o *bounded context* de “Gestão de Recursos” identificado na arquitetura, mantendo autonomia sobre os dados de viaturas e disponibilidades através do padrão *Database per Service*. A implementação baseia-

se numa arquitetura *REST*, com um *controller* único que expõe os *endpoints* para a gestão de recursos de automóveis, conforme especificado no Apêndice D. A interface permite operações de consulta individual e coletiva de viaturas, registo de novos veículos, atualização de dados existentes e remoção de viaturas sem histórico associado. A separação deste microserviço permite a evolução independente das funcionalidades de gestão de recursos, facilitando futuras extensões, tais como controlo de manutenções e gestão de combustível, sem impacto nos restantes componentes da solução.

5.1.6. NotifyFlow API

O NotifyFlow API constitui um microserviço para o envio de notificações por email, responsável por coordenar a comunicação externa com os utilizadores durante o ciclo de vida dos processos de autorização. Este microserviço materializa o *bounded context* de "Comunicação e Notificações" identificado na arquitetura, mantendo autonomia sobre o histórico de comunicações e configurações de entrega através de uma base de dados MySQL dedicada. A implementação baseia-se na biblioteca MailKit para comunicação via protocolo *Simple* Mail Transfer Protocol (SMTP), encapsulada na classe `SendEmail`, que utiliza o padrão *Options* para configuração através de `SmtpSettings` (servidor, porta, credenciais, remetente). O serviço suporta autenticação *StartTLS* e validação de certificados personalizável, permitindo integração com diferentes fornecedores de email corporativo. A construção das mensagens utiliza `MimeMessage` com suporte a conteúdo HTML, que permite formatação nas notificações enviadas.

O *controller* expõe funcionalidades para envio de notificações por email e consulta do histórico de comunicações. A operação de envio aceita objetos `EmailRequest` com validação de campos obrigatórios, processa o envio através do serviço `SendEmail` e regista a operação na tabela `EmailLog` com timestamp UTC para efeitos de auditoria. As operações de consulta permitem aceder ao histórico de notificações, suportando tanto a listagem completa como a filtragem por endereço de email específico, conforme especificado no Apêndice E.

O sistema implementa *logging* estruturado para rastreabilidade das operações de envio, regista sucessos e falhas com códigos HTTP apropriados. A integração com outros microserviços é realizada através de comunicação assíncrona via RabbitMQ, consumindo mensagens da fila `NotificationQueue` publicadas pela API *Demo* durante a execução de *external tasks* do tipo "Enviar". Esta abordagem assíncrona permite que o *workflow* principal continue independentemente

da latência de entrega de emails, mantendo a performance do processo de negócio enquanto entrega a notificação dos utilizadores sobre aprovações, rejeições ou necessidade de ação adicional.

5.2. Implementação de Padrões de Integração

Esta seção descreve a implementação de padrões de integração aplicados a vários microserviços da solução, incluindo API Gateway, comunicação assíncrona e monitorização, descritos nas subseções seguintes.

5.2.1. API Gateway e Circuit Breaker com Ocelot

O API Gateway Pattern foi implementado com recurso à biblioteca *Ocelot*, integrada no ecossistema .NET, com o objetivo de centralizar o acesso aos microserviços. A abordagem adotada recorre a configuração declarativa em ficheiros JSON. A decisão de centralizar o acesso justifica-se pela simplificação da arquitetura das aplicações cliente, pela redução da duplicação de lógica de comunicação e pela facilitação da manutenção através de uma interface de acesso uniformizada.

A implementação baseia-se na definição de rotas que mapeiam *endpoints* externos para serviços internos, recorrendo aos padrões *UpstreamPathTemplate* e *DownstreamPathTemplate* para definir as rotas com base em URL e métodos HTTP. São suportados três tipos de rotas: rotas específicas para operações individuais; rotas parametrizadas para recursos identificados por ID; e rotas genéricas com suporte a *wildcard* (ex.: `/gateway/pdf/{everything}`), o que permite maior flexibilidade futura. A configuração `GlobalConfiguration` define o `BaseUrl` único, que funciona como *reverse proxy* e redireciona pedidos externos para os microserviços internos. Esta centralização oculta a topologia da rede e permite que as aplicações acedam a todos os serviços através de um ponto único, independentemente da localização física de cada microserviço.

O Circuit Breaker Pattern é integrado através da biblioteca *Polly*, registada via `.AddPolly()` no ficheiro `Program.cs`. A configuração é especificada em `QoSOptions` no ficheiro `ocelot.json`, contemplando três níveis de criticidade definidos em função da importância operacional dos serviços. Para o Identity Server, considerado crítico para autenticação, o circuito abre após cinco exceções consecutivas, mantendo-se em estado aberto durante 10000 ms e cada operação tem um tempo máximo de execução de 15000 ms. Um exemplo simplificado de configuração encontra-se ilustrado na Figura 17.

```
"QoSOptions": {
   "ExceptionsAllowedBeforeBreaking": 5,   // número de exceções consecutivas permitidas
    "DurationOfBreak": 10000,               // duração do circuito aberto (ms)
    "TimeoutValue": 15000                // tempo máximo de execução por operação (ms)
}
```

Figura 17 - Configuração QoS para serviço crítico

Os restantes serviços seguem configurações ajustadas às suas características, os *workflows* BPMN do Demo API têm *timeouts* superiores (25000 ms) devido à natureza do processamento de tarefas *Zeebe*, enquanto o PDF Generator Service utiliza 30000 ms de *timeout*, considerando a complexidade da geração de documentos.

No total, a implementação abrange 15 rotas distribuídas pelos microserviços, aplica proteção contra falhas em cascata através do princípio *fail-fast*, que permite a recuperação automática assim que os serviços retomam disponibilidade. Esta estratégia confere resiliência global à aplicação, mesmo perante indisponibilidades temporárias de componentes individuais.

5.2.2. Event-Driven Pattern com RabbitMQ

A implementação do *Event-Driven Pattern* utiliza RabbitMQ como *message broker* para coordenar comunicação assíncrona entre microserviços, permitindo desacoplamento temporal e lógico entre componentes da solução. Esta abordagem resolve dependências síncronas que poderiam comprometer a disponibilidade global do sistema.

A estratégia utiliza filas de mensagens dedicadas para diferentes tipos de eventos. O processo de criação de utilizadores exemplifica esta abordagem, onde o *Users API* publica eventos que o *Identity Server* consome de forma assíncrona, conforme apresentado na Figura 18.

```
private async Task SendMessageToRabbitMQ(UserDTO userDto)
{
    var connection = _rabbitConnectionFactory.CreateConnection();
    using (var channel = connection.CreateModel())
    {
        channel.QueueDeclare("QueueIdentity ", exclusive: false);
        var message = JsonSerializer.Serialize(userDto);
        var body = Encoding.UTF8.GetBytes(message);
       channel.BasicPublish(exchange: "", routingKey: " QueueIdentity ", body: body);
    }
}
```

Figura 18 - Exemplo simplificado RabbitMQ de publicação de evento para criação de utilizador

O consumidor no *Identity Server* processa eventos de forma contínua, conforme exemplificado na Figura 19.

```
consumer.Received += (model, eventArgs) =>
{
    var body = eventArgs.Body.ToArray();
    var message = Encoding.UTF8.GetString(body);
    var userDto = JsonSerializer.Deserialize<UserDTO>(message);
    _userCreationService.CreateUserAsync(userDto, userManager, roleManager);
};
channel.BasicConsume(queue: " QueueIdentity ", autoAck: true, consumer: consumer);
```

Figura 19 - Exemplo simplificado RabbitMQ de consumo de eventos para criação de utilizador

A implementação evidencia os benefícios decorrentes da adoção deste padrão, o Users API deixa de depender da confirmação imediata do *Identity Server*, garantindo que os eventos permanecem na fila mesmo perante indisponibilidades temporárias dos serviços, o que possibilita uma escalabilidade independente entre componentes. Para além disso, o padrão constitui uma base estruturante que pode ser explorada em futuras extensões, nomeadamente no processamento de eventos de maior complexidade.

5.2.3. Health Check Pattern

A implementação do *Health Check Pattern* utiliza *endpoints /health* dedicados que devolve informação estruturada sobre o estado operacional dos microserviços. Esta abordagem permite monitorização automatizada e integração com sistemas de orquestração. Para esse efeito é apresentado um exemplo na Figura 20.

```
ResponseWriter = async (context, report) =>
{
    context.Response.ContentType = "application/json";
    var response = new
    {
        status = report.Status.ToString(),
        checks = report.Entries.Select(x => new
        {
            name = x.Key,
            status = x.Value.Status.ToString(),
            description = x.Value.Description,
            duration = x.Value.Duration.TotalMilliseconds
        }),
        totalDuration = report.TotalDuration.TotalMilliseconds
    };
                                                                        await
context.Response.WriteAsync(System.Text.Json.JsonSerializer.Serialize(response));
}
```

Figura 20 - Exmplo implementação do ResponseWriter para health checks

A resposta segue uma estrutura padronizada que agrega o estado global e as verificações individuais. Esta verificação inclui o nome, o estado (Healthy, Degraded ou Unhealthy), uma descrição textual da memória e o tempo de execução em milissegundos. A Figura 21 mostra um exemplo da saída gerada pelo sistema.

```
01 {
02     "status": "Healthy",
03     "checks": [{ "name": "memory",
04             "status": "Healthy",
05             "description": "Memória OK: 4MB",
06             "duration": 10.25
07         }], "totalDuration": 37.687}
```

Figura 21 - Exemplo de resposta JSON do health check

No exemplo apresentado, é evidenciada a monitorização da memória, mas a mesma abordagem pode ser aplicada a outros componentes. Este padrão estabelece uma base extensível para integração com sistemas de monitorização e alertas, bem como para estratégias de *load balancing*, permitindo acompanhar continuamente a disponibilidade e desempenho dos microserviços em produção.

5.3. Implementação de *Blue-Green Deployment* com *Docker*

A implementação do padrão *Blue-Green Deployment* foi adaptada para ambiente de desenvolvimento local através de uma variação *Red-Black* que mantém os benefícios principais de ambientes isolados e *rollback* rápido. A abordagem seguida privilegia a simplicidade de configuração sem necessidade de alterações nos pontos de acesso existentes, optando por uma transição sequencial em vez da clássica do Blue-Green. Esta adaptação justifica-se pela necessidade de manter retrocompatibilidade com a configuração atual de desenvolvimento. Em ambiente de testes futuro, a implementação clássica será testada com o K3s.

A implementação foi realizada através de ficheiros *Docker-compose* separados para cada ambiente, complementada por um *script* PowerShell que automatiza o processo de transição. Foram criados os ficheiros *Docker*-compose-blue.yml e *Docker*-compose-green.yml. A configuração dos ambientes mantém as portas externas, garantindo que não são necessárias alterações no desenvolvimento.

A Figura 22 ilustra a arquitetura completa da implementação *Blue-Green Deployment*, demonstra como os dois ambientes idênticos (Blue e Green) coexistem, sendo que apenas um permanece ativo num dado momento.

Figura 22 - Adaptação da arquitetura Blue-Green Deployment

Conforme representado na Figura 22, cada ambiente possui uma instância completa de todos os microserviços. A transição entre ambientes é efetuada através de um *script* PowerShell automatizado que gere todo o processo entra 20 a 50 segundos e inclui a paragem controlada do ambiente ativo e inicialização do ambiente de destino.

Os serviços partilhados, nomeadamente a base de dados MySQL e o sistema de monitorização *Grafana*, mantêm-se ativos independentemente do ambiente, garantindo persistência de dados e configurações entre transições. Esta estratégia permite que ambos os ambientes acedam aos mesmos recursos partilhados através da rede *Docker monitoring,* o que elimina a necessidade de migração de dados durante as transições.

Comparativamente ao *deployment* tradicional, onde cada atualização requer paragem completa do sistema com *downtime* potencialmente extenso, esta implementação reduz significativamente o tempo de indisponibilidade. Esta implementação comprova a viabilidade de adaptar padrões *Enterprise* para desenvolvimento local, o que proporciona uma transição natural para ambientes de produção onde ferramentas como *load balancers* ou orquestradores de *containers* podem implementar o padrão clássico *Blue-Green*.

5.4. Migração dos Microserviços para *Docker*

Este tópico documenta a migração de microserviços de ambiente nativo para *containers Docker*, apresenta o processo técnico de containerização, os desafios de configuração e autenticação encontrados.

5.4.1. Implementação da Containerização e Configuração *Multi-Stage*

A migração dos microserviços para *Docker* seguiu uma abordagem incremental. Os serviços mais simples e com maior independência funcional foram os primeiros a serem migrados. A implementação utilizou *Docker*files multi-stage para otimização das imagens finais, separando as fases de *build* e *runtime*. O processo *multi-stage* permitiu utilizar uma imagem com SDK completo para compilação na primeira fase, copiando apenas os binários compilados para uma imagem *runtime* mais leve na segunda fase através de `COPY --from=build`. Esta abordagem reduziu significativamente o tamanho das imagens finais, excluindo dependências de desenvolvimento através de configuração `.Dockerignore` que eliminou `bin/, obj/, logs` e ficheiros de configuração desnecessários.

A estratégia de migração centrou-se na manutenção de retrocompatibilidade durante todo o processo. O *Docker*`-compose.yml` implementou uma rede personalizada *monitoring* para isolar a comunicação entre serviços, com mapeamento de portas específico: pdf-generator-service (7281:8080), identity-server (7218:7218), notify-flow-api (7168:8081), demo-api (7283:7283), users-api(7039:7039) api-gateway (5069:5069) e vehicles-api (7282:7282).

O API *Gateway Ocelot* foi configurado, conforme um pequeno exemplo na Figura 23, para encaminhar pedidos tanto para versões locais como para versões containerizadas de diversos serviços. Nesta configuração, o parâmetro `DownstreamHostAndPorts` define o endereço e a porta do serviço real, enquanto `UpstreamPathTemplate` indica o caminho utilizado pelo cliente para aceder ao serviço através do *gateway*. Durante a migração, cada serviço manteve duas rotas ativas: uma para a execução local, usando o endereço *localhost* e a porta exposta no *host* (por exemplo, 7281), e outra para a execução em container, usando o nome do serviço no *Docker* e a porta interna do container (por exemplo, pdfservice:8080). Esta diferença ocorre porque, no ambiente *Docker*, o Ocelot também é executado em container e comunica diretamente pela rede interna, sem necessidade de usar o mapeamento de portas do *host*. Esta abordagem permitiu que o Ocelot encaminhasse automaticamente os pedidos para a instância disponível, evitando interrupções e possibilitando *rollback* imediato se necessário.

```
// Execução local
{
  "DownstreamHostAndPorts": [
    { "Host": "localhost", "Port": 7281 }
  ],
  "UpstreamPathTemplate": "/gateway/generatepdf"
},
// Execução em container Docker
{
  "DownstreamHostAndPorts": [
    { "Host": "pdfservice", "Port": 8080 }
  ],
  "UpstreamPathTemplate": "/gateway/pdf/{everything}"
}
```

Figura 23 - Exemplo simplificado de configuração Ocelot

Esta abordagem permitiu retrocompatibilidade durante todo o processo de migração através de *deployment* mutuamente exclusivo. Cada serviço foi migrado individualmente enquanto os restantes continuaram a operar nativamente, evitando conflitos de portas e recursos. Por exemplo, durante a migração do identity-server para container, a versão nativa foi terminada para evitar *binding conflicts* na porta 7218, enquanto os outros serviços (pdf-generator-service, notify-flow-api) mantiveram-se operacionais em *localhost*. A configuração *dual* do Ocelot permitiu definir múltiplos *endpoints* alternativos para cada serviço, garantindo *failover* imediato em caso de problemas, sem necessidade de reconfiguração do *gateway*.

5.4.2. Desafios de Configuração e Resolução de Problemas de Autenticação

A migração apresentou desafios significativos na adaptação de configurações para o ambiente containerizado. Por exemplo, no Identity Server foi necessário alterar a *connection string*, substituindo o servidor *localhost* por db e especificar a porta 3306, de forma a direcionar as conexões para o container da base de dados em vez do host.

A gestão de certificados TLS constituiu outro desafio técnico, o que exigiu o mapeamento dos ficheiros de certificado dentro dos *containers* para que todos os serviços possam aceder aos mesmos de forma consistente. O desafio mais complexo envolveu o erro 401 *Unauthorized* sistemáticos devido a incompatibilidades nos *tokens* JWT. Por exemplo, *tokens* gerados para o *endpoint localhost* falhavam validação em serviços configurados para o *hostname* do *container identity-server*. Os *claims aud* (audience) e *iss* (issuer) dos *tokens* não correspondiam aos *endpoints* esperados pelos serviços containerizados. A resolução exigiu configuração de múltiplos valores válidos no *ValidIssuer* e

ValidAudience dos serviços consumidores, permitindo aceitar *tokens* de ambos os *endpoints* de forma controlada durante o período de transição.

5.5. Migração e Avaliação de Microserviços de *Docker* para K3s (*Kubernetes*)

Esta secção apresenta a migração de microserviços de *Docker* para K3s, documentando o processo técnico, os desafios de implementação e a avaliação experimental de desempenho entre as duas plataformas de orquestração.

5.5.1. Contexto e Processo de Migração

Após a migração prévia de todos os microserviços para *Docker*, este trabalho foca uma abordagem híbrida para K3s. Foi selecionado um microserviço específico, o serviço de geração de PDF, como caso de estudo para validação da viabilidade técnica da migração. O serviço executava com um certificado HTTPS autoassinado configurado através de ficheiros PFX, enquanto o *gateway* redirecionava pedidos para o serviço através de uma porta específica. A aplicação utilizava volumes *Docker* para persistência de PDFs gerados, permitindo acesso direto a partir do *host*.
A migração manteve o *Docker* como *runtime* de containers e adicionou a orquestração *Kubernetes* através do K3s. O comando utilizado para criação do *cluster* foi: `k3d cluster create proj-cluster --volume "origem:/destino@all"`

O processo exigiu a criação de manifestos YAML para definir os recursos K3s necessários. O *Deployment* especificou a configuração dos *pods*, incluindo variáveis de ambiente para configuração conforme Apêndice F. O *Service (*Apêndice G) foi configurado para exposição interna através de *ClusterIP*, mantendo compatibilidade com a aplicação existente.

A configuração de rede utilizou *port-forwarding* através do comando `kubectl port-forward` para mapear portas locais às portas dos serviços, preservando a compatibilidade com o *gateway* Ocelot existente. Esta estratégia permitiu manter a interface de acesso sem necessidade de alterações na camada de roteamento.
A gestão de certificados TLS no K3s apresentou desafios no mapeamento de volumes entre o sistema *host* Windows e os *containers* Linux. Os certificados necessitavam de estar acessíveis nos containers através de volumes dedicados, o que exigiu a configuração de variáveis de ambiente específicas do Kestrel (servidor web) para garantir o funcionamento correto do HTTPS, sobrepondo as definições existentes nos ficheiros da aplicação. Inicialmente os caminhos definidos nos *Deployments* não correspondiam aos volumes mapeados aquando da criação do *cluster*, originando erros nos *logs* dos

pods o que obrigou a várias recriações do *cluster* para corrigir o problema. Adicionalmente, verificaram-se conflitos entre as definições de HTTPS nos ficheiros de configuração e nas variáveis de ambiente, sendo necessário ajustar estas últimas para sobrepor as configurações da aplicação.

5.5.2. Implementação Multi-Node

A segunda fase envolveu a implementação de um *cluster* multi-*node* com um *control plane* e dois *worker nodes para isso foi utilizado o comando* `k3d cluster create proj-cluster-multi --agents 2 \ --volume "origem:/destino@all".` O teste de escalabilidade foi realizado através do comando: `kubectl scale deployment pdf-generator-deployment --replicas=3`

Esta operação demonstrou a distribuição automática dos *pods* pelos *nodes* disponíveis, resultando numa distribuição onde cada *node* executava um *pod* com endereços IP distintos da rede interna do *cluster*.

A validação do *load balancing* foi efetuada através de múltiplos pedidos HTTP consecutivos, onde o *Service K3s* distribuiu automaticamente os pedidos pelos *pods* disponíveis sem necessidade de configuração adicional. Esta funcionalidade demonstra as capacidades nativas de balanceamento de carga do K3s em ambientes distribuídos. A migração estabeleceu uma infraestrutura com capacidades de *self-healing*, onde *pods* falhados são automaticamente substituídos, e *rolling updates* que permitem atualizações sem interrupção de serviço.

A abordagem híbrida demonstrou viabilidade prática, permitindo que o *gateway* Ocelot continuasse operacional sem alterações de configuração, acedendo ao serviço migrado para K3s através da mesma interface. Esta compatibilidade valida estratégias de migração incremental, onde diferentes microserviços podem operar em tecnologias de orquestração distintas dentro do mesmo ecossistema.

5.5.3. Validação Experimental e Análise Comparativa

Para validar as implicações da migração, foram realizados testes comparativos entre três configurações: container *Docker standalone*, *cluster* K3s com um único *pod*, e *cluster* com três *pods* distribuídos. A metodologia utilizou um script (Apêndice H) que executou vinte requisições HTTP sequenciais para cada configuração, com payloads JSON. O intervalo de 250 milissegundos entre requisições foi determinado após cinco execuções preliminares, garantindo que não ocorresse saturação excessiva, e é um valor parametrizável, podendo ser ajustado conforme a necessidade do teste.

Os resultados apresentados na Tabela 4 revelaram diferenças significativas de performance entre as configurações. O script foi executado dez vezes, enviando vinte requisições sequenciais em cada execução, garantindo um número semelhante de testes em todos os cenários. O container *Docker* direto demonstrou performance superior, processou as requisições com tempo médio de resposta de 214 milissegundos (variando entre 128 ms e 458 ms) e *throughput* (capacidade de processamento expressa em requisições por segundo) de 2,12 requisições por segundo. A configuração com um único *pod* no K3s apresentou tempo médio de 903 milissegundos e *throughput* de 0,86 requisições por segundo, enquanto a configuração distribuída com três *pods* registou tempo médio de 1370 milissegundos e *throughput* de 0,61 requisições por segundo. Todas as configurações mantiveram taxa de sucesso de 100%. Para assegurar comparabilidade, os testes foram realizados no mesmo ambiente base, utilizando o mesmo hardware, versão de sistema operativo e configuração de rede, e o intervalo entre requisições foi mantido constante.

CONFIGURAÇÃO	**TEMPO MÉDIO**	**TEMPO MÍNIMO**	**TEMPO MÁXIMO**	***THROUGHPUT***	**TAXA SUCESSO**
DOCKER STANDALONE	214ms	128ms	458ms	2,12 req/s	100%
K3S (1 *POD*)	903ms	555ms	1313ms	0,86 req/s	100%
K3S (3 *PODS*)	1370ms	946ms	1892ms	0,61 req/s	100%

Tabela 4 - Comparação de performance entre configurações Docker e K3s

A análise dos resultados evidencia que o *Docker standalone* é aproximadamente quatro vezes mais rápido que a configuração K3s com um *pod* e seis vezes mais rápido que a configuração distribuída. Esta diferença resulta da ausência do *overhead* (sobrecarga computacional adicional) de orquestração presente no K3s.

O comportamento onde um *pod* único supera três *pods* distribuídos explica-se pela natureza sequencial das requisições testadas. Para cargas sequenciais, a distribuição introduz *overhead* adicional sem benefícios de paralelização. A distribuição da carga entre os três *pods* revelou comportamento assimétrico, com concentração de requisições num *pod* específico, característica dos algoritmos de balanceamento de carga que priorizam eficiência sobre distribuição perfeitamente equitativa.

Estes resultados, embora evidenciem *overhead* de performance no K3s, devem ser interpretados considerando as limitações dos testes sequenciais realizados. A diferença de performance observada ilustra o compromisso entre simplicidade operacional e capacidades de orquestração.
Para cargas de trabalho simples e ambientes estáveis, o *Docker standalone* demonstra clara superioridade em performance. Contudo, para aplicações que necessitem de escalamento automático, alta disponibilidade ou gestão complexa de falhas, o *overhead* do K3s pode ser justificado pelos benefícios operacionais.

Os testes validaram o *load balancing* automático, escalamento dinâmico e manutenção de 100% de disponibilidade, confirmando a funcionalidade da migração. Futuras investigações devem incluir cenários de carga concorrente e falhas simuladas para avaliar adequadamente o valor da capacidade do K3s.

A implementação piloto estabelece uma base para decisões informadas sobre migração gradual, demonstrando que a escolha entre *Docker* e K3s deve considerar não apenas performance bruta, mas os requisitos específicos de cada microserviço e o contexto operacional da aplicação.

5.6. Frontend

A camada de apresentação implementa uma aplicação web Angular que coordena interações entre utilizadores e microserviços através do *API Gateway* centralizado. Esta secção aborda a implementação da camada de apresentação, o fluxo completo do processo de requisição de viatura, e os módulos de gestão de recursos e configurações, demonstrando a integração entre interface de utilizador e execução de *workflows* BPMN.

5.6.1. Implementação da Camada de Apresentação

A interface de utilizador é implementada em Angular 16 como uma *Single Page Application* (SPA) que comunica exclusivamente através do *API Gateway.* A aplicação adota uma arquitetura baseada em módulos de serviço com injeção de dependências e *guards* de rota, asseguram controlo de acesso. O sistema de autenticação implementa módulos de interceção HTTP que processam *tokens* JWT, utilizando *localStorage* para persistência de sessão e disponibilização dos dados de utilizador. A gestão de rotas utiliza quatro tipos de *guards*: `AuthGuard` para autenticação básica, `AdminGuard` para funcionalidades de gestão de administradores, `ClientGuard` para operações com as permissões

minimas, e `decisionGuard` para permissões que permitem a tomada de decisão. Esta segmentação permite controlo sobre funcionalidades por perfil de utilizador.

O sistema de alertas centraliza notificações através do `AlertService` que encapsula `SweetAlert2`, que oferece métodos padronizados de alerta e notificação incluindo diálogos modais, notificações *toast* temporárias, e confirmações com *callbacks* personalizados. Esta abordagem permite consistência visual e comportamental nas interações. A arquitetura de componentes privilegia a reutilização de código, recorre à parametrização de rotas para adaptar a mesma interface a diferentes operações. Por exemplo, o `CarFormComponent` é utilizado tanto para a criação como para a edição de viaturas.

A validação de formulários implementa lógica condicional no *frontend* que complementa validação *backend*. A implementação de formulários utiliza *Reactive Forms* do Angular com `FormBuilder` para construção declarativa de controlos validados. Os formulários implementam estados condicionais através de propriedades *disabled* para campos pré-preenchidos e aplicam validações síncronas (`Validators.required`) com *feedback* visual imediato através de diretivas condicionais. A validação de erro apresenta mensagens contextualizadas baseadas no estado dos controlos, proporciona orientação específica ao utilizador durante o preenchimento.

A gestão do ciclo de vida dos componentes implementa inicialização controlada através de `OnInit` e gestão de subscrições `RxJS`. O código está organizado em classes de serviço para a lógica de negócio, componentes para a apresentação (HTML/CSS) e interfaces para definir a estrutura dos dados, favorecendo a manutenção.

5.6.2. Fluxo Completo do Processo de Requisição de Viatura

O processo inicia-se com autenticação do utilizador através da interface de *login*, conforme mostrado na Figura 24 onde as credenciais são validadas pelo *Identity Server*. A interface implementa validação visual com indicadores de erro, redirecionando para o *dashboard* principal após validação bem-sucedida.

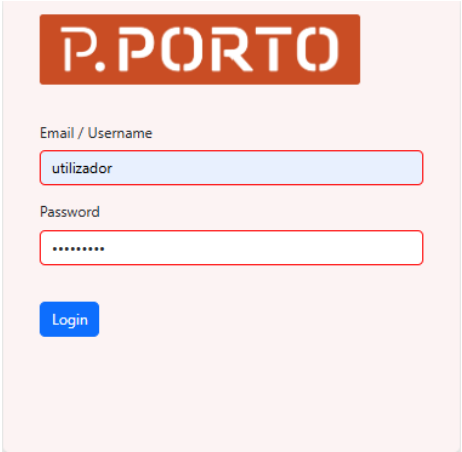


Figura 24 - Interface de login

A submissão de requisição corresponde ao elemento BPMN `Form_SubmetePedido` e apresenta um formulário estruturado em que os campos Nome Completo e Categoria Profissional são populados automaticamente a partir dos dados obtidos do microserviço *Users*. Como ilustrado na Figura 25, o utilizador seleciona o tipo de autorização através de *checkboxes* (A – Conduzir viatura de serviços gerais, B – Automóvel próprio, C – Automóvel próprio nº4), define a descrição do serviço, a rota e o período de utilização através de controlos de data/hora com validação temporal.

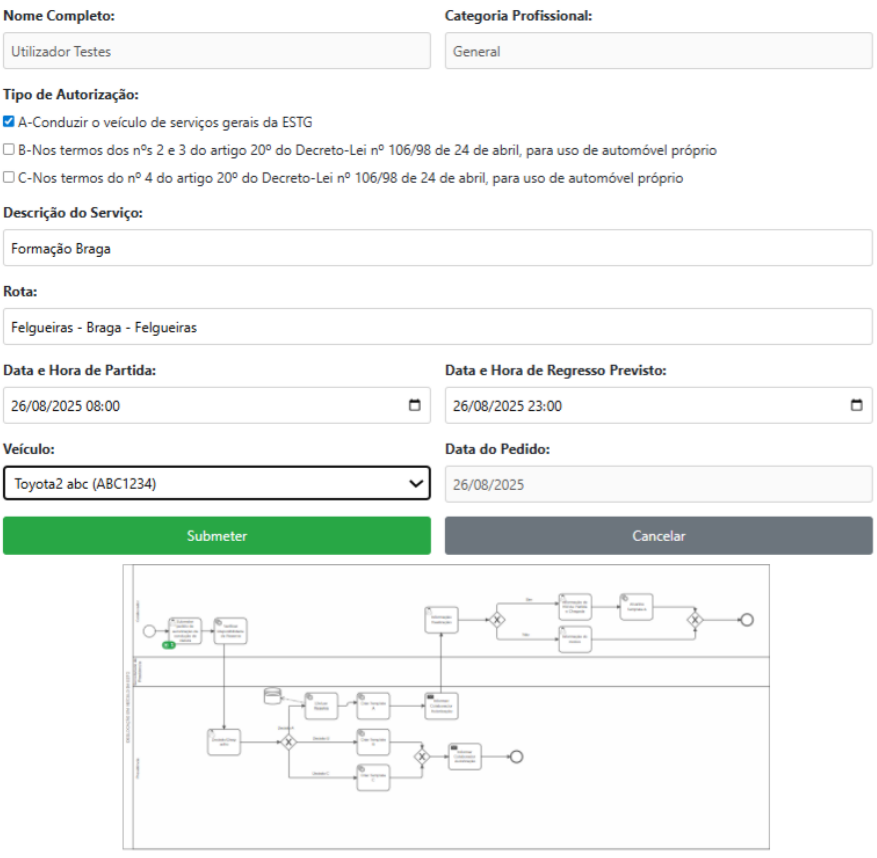


Figura 25 - Formulário de submissão de requisição

A seleção de viatura é realizada através de um *dropdown* populado dinamicamente pelo microserviço *Vehicles API*. A submissão inicia uma nova instância do processo BPMN através do motor *Zeebe*.

Após a submissão da requisição, o fluxo do processo passa para os utilizadores com permissões de decisão. Nessa fase, a interface apresenta uma lista de requisições pendentes organizada por número de reserva, requerente, serviço, datas, viatura e estado da decisão. Esta tabela permite pesquisa em todos os campos, o que facilita a localização de pedidos específicos. Como mostrado na Figura 26, o botão *Decisão* permite o acesso individual a cada requisição para processamento. Após a decisão, o estado da requisição é atualizado automaticamente de pendente para concluído, sendo a entrada transferida para o histórico, que permanece acessível para consulta a qualquer momento.

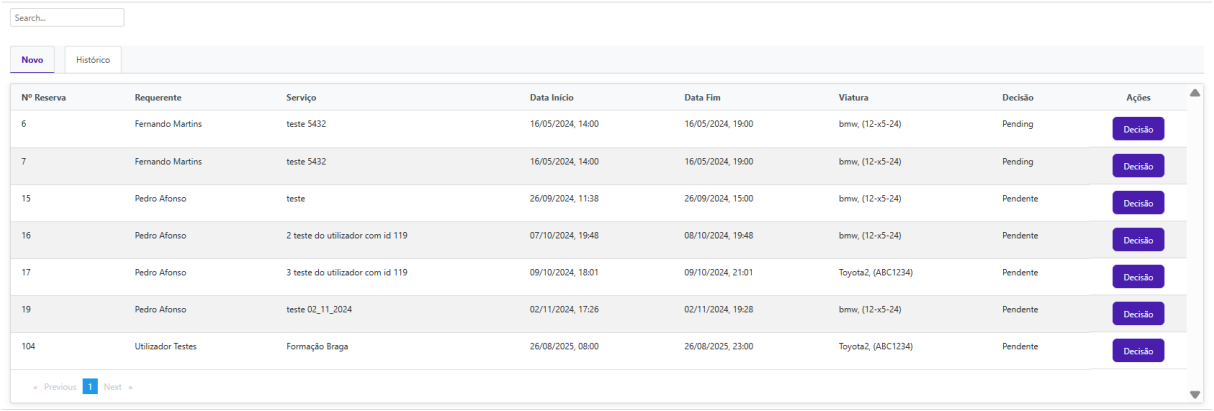


Figura 26 - Lista de requisições pendentes

O ecrã de decisão corresponde ao elemento BPMN `decisão` conforme ilustrado na Figura 27 apresenta, no painel esquerdo, a visualização completa dos dados da requisição submetida. Em paralelo, o painel direito disponibiliza opções de decisão através de um *dropdown* (Deferido/Indeferido) e campo para comentário justificativo. A submissão desta informação atualiza as variáveis do processo e aciona um *gateway* exclusivo baseado no tipo de autorização, o qual gera automaticamente documentos PDF a partir dos *templates* definidos conforme o exemplo de um documento gerado no Apêndice C. Posteriormente é despoletado um e-mail a informar da decisão.

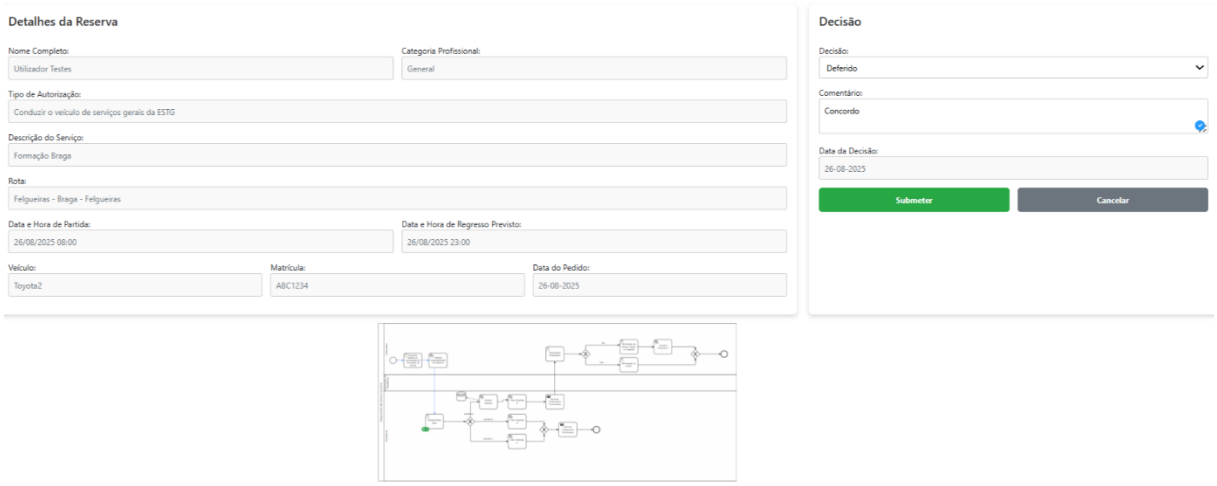


Figura 27 - Interface de decisão/despacho

No caso de ser aprovado e o *template* ser do tipo A, o sistema avança para a fase de realização, representada pelo elemento BPMN `InfoRealizacao`. Nesta etapa, a interface inicial apresenta um formulário em que o utilizador confirma a execução da deslocação e regista os quilómetros de partida, dando início ao controlo da utilização da viatura institucional, conforme mostrado na Figura 28.

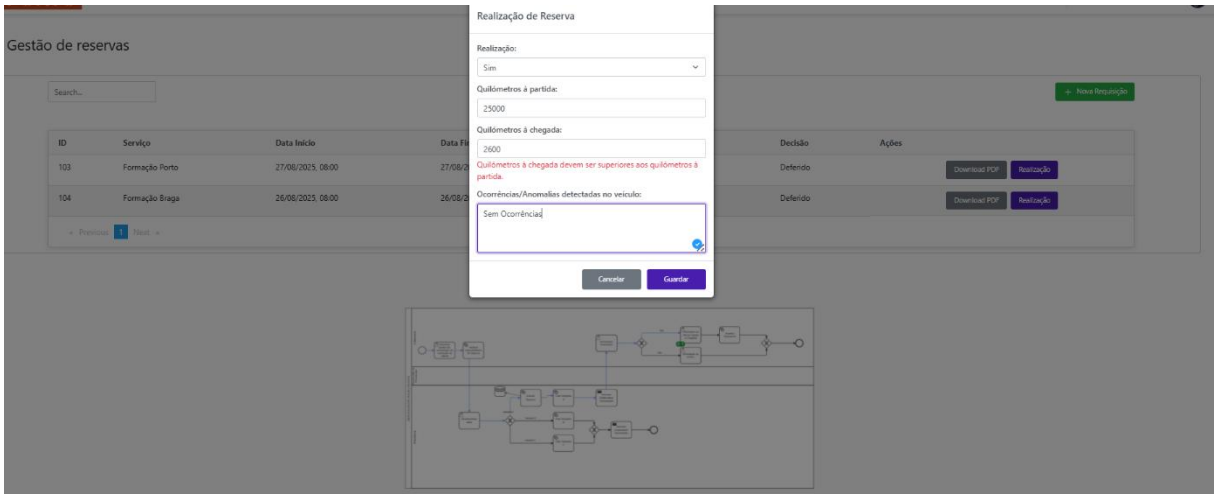


Figura 28 - Realização de reserva - quilómetros iniciais

Na conclusão da realização, o sistema solicita a introdução dos quilómetros de chegada, impondo a validação de valores superiores aos de partida. Caso contrário, são exibidas mensagens de erro contextualizadas. Nesta mesma fase, o campo destinado ao registo de ocorrências ou anomalias permite documentar eventuais problemas detetados durante a utilização. Este procedimento corresponde ao elemento BPMN `infokm` e aciona a atualização do documento gerado através da `external task AtualizaTemplate_A`, como ilustrado na Figura 29.

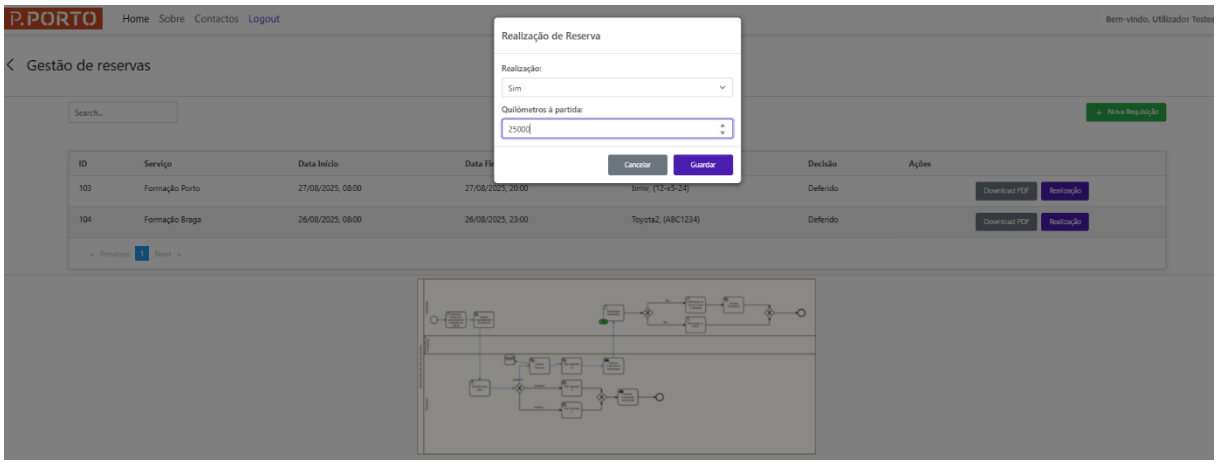


Figura 29 - Realização de reserva - quilómetros finais e validação

Por fim, o diagrama BPMN completo, apresentado na Figura 30, sintetiza a progressão do processo, desde a submissão inicial até à conclusão da realização, destaca a separação de responsabilidades entre as *lanes* do Colaborador e da Presidência. Esta representação visual permite a rastreabilidade integral do processo executado.

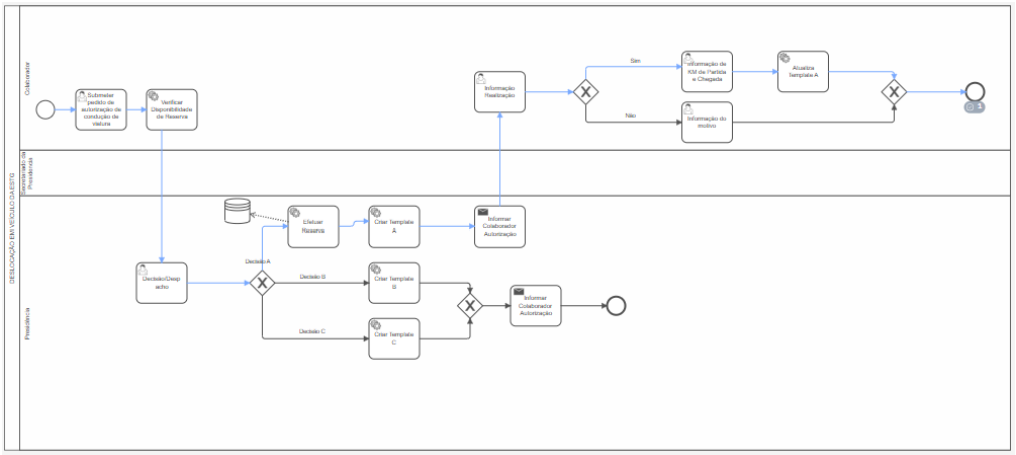


Figura 30 - Diagrama BPMN completo do processo

De forma global, a integração entre a interface e o processo BPMN estabelece uma correspondência direta entre elementos de *workflow* e funcionalidades da aplicação. Assim, o sistema mantém a sincronização entre o estado visual e a execução através de atualizações coordenadas por microserviços, assegurando validação distribuída entre camadas de apresentação e lógica de negócio, o que proporciona simultaneamente *feedback* imediato e consistência operacional.

5.6.3. Gestão de Recursos e Configurações (Interface Principal)

A interface principal, como mostrado na Figura 31, é adaptativa baseada em permissões de utilizador. As funcionalidades estão organizadas hierarquicamente em módulos. Os utilizadores com perfil administrativo têm acesso às funcionalidades de Gestão de Utilizadores, Minha Conta, Viatura de Serviço, Reservar Viatura, Parecer/Decisão de Reservas, Configurações LDAP e Dados Estatísticos. Os utilizadores com permissões restritas visualizam apenas os módulos compatíveis com o respetivo perfil.

Figura 31 - Dashboard principal com permissões administrativas

A gestão de configurações LDAP permite a administração de múltiplas instâncias de *Active Directory*, como apresentado na Figura 32. Esta funcionalidade é suportada por um formulário estruturado que define o URL LDAP, as credenciais de ligação, a *Base DN (Distinguished Name)*, o domínio e o estado de ativação. A interface apresenta uma listagem tabular das configurações existentes, acompanhada de indicadores visuais de estado (Ativo/Inativo) e de ações de edição individual. O módulo suporta integração híbrida entre autenticação local e federada, permitindo a validação de credenciais em múltiplos domínios organizacionais.

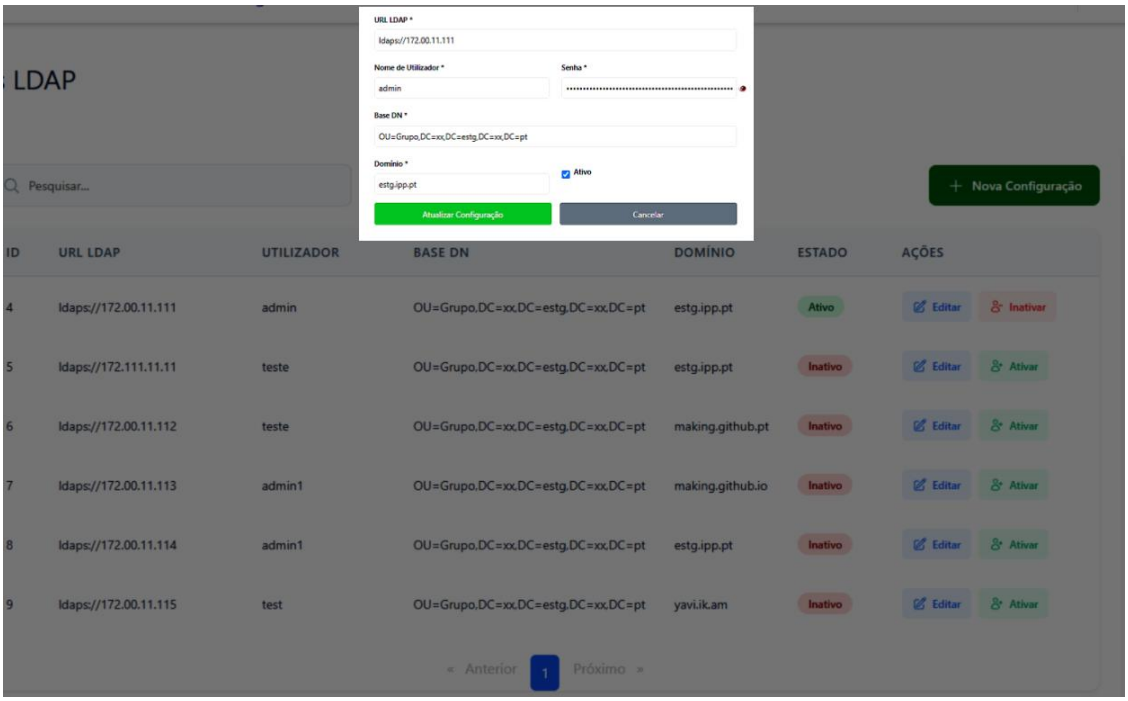


Figura 32 - Configuração e gestão LDAP

O módulo de gestão de utilizadores implementa operações CRUD através de uma interface tabular, como ilustrado na Figura 33. Esta interface apresenta dados estruturados (ID, Primeiro Nome, Último Nome, Email, Categoria Profissional, Nome de Utilizador, Perfil) e ações contextuais associadas a cada registo. O formulário de edição permite a atualização de propriedades individuais, incluindo dados pessoais, categoria profissional, tipo de utilizador. Os campos de *password* são apresentados de forma ofuscada. A funcionalidade suporta a criação direta de contas locais e a sincronização com dados provenientes de autenticação LDAP.

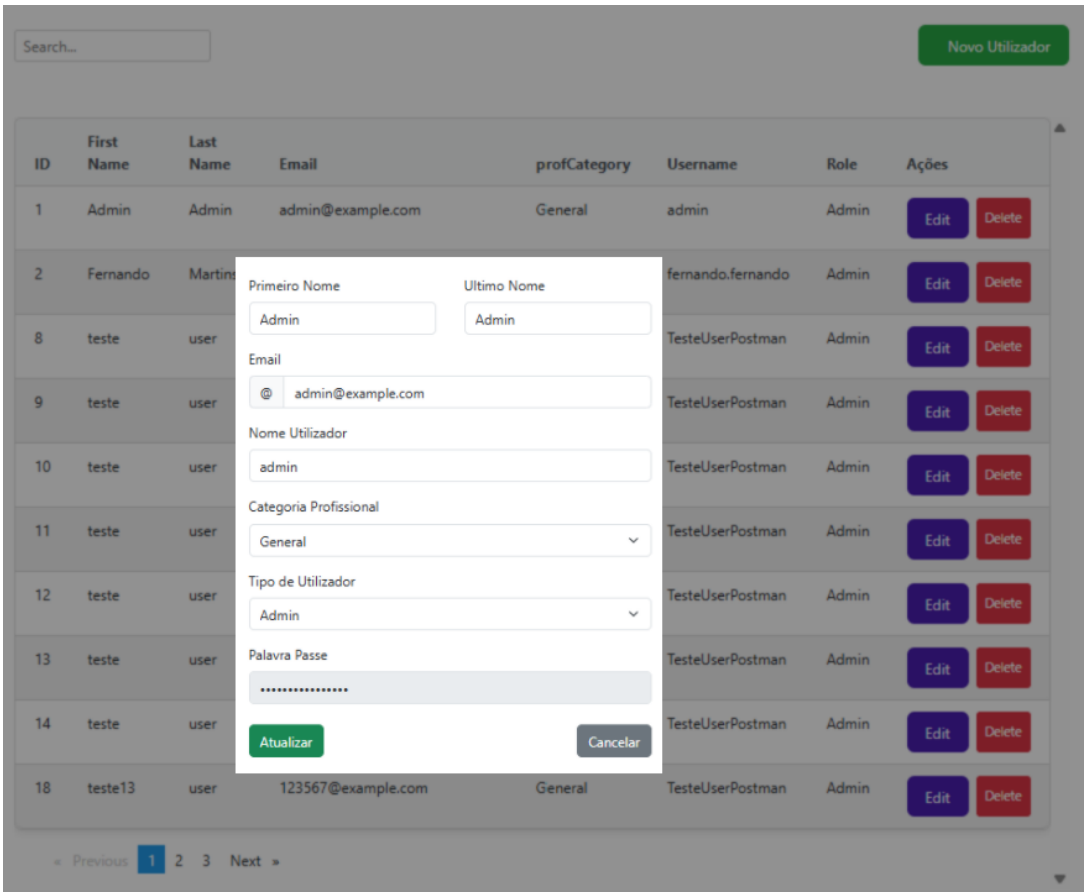


Figura 33 - Gestão de utilizadores e formulário de edição

A administração de viaturas gere o catálogo de recursos através de uma interface que controla propriedades como marca, matrícula, modelo e tipo de combustível, como mostrado na Figura 34. O tipo de combustível é selecionado através de *dropdown* estruturado. A funcionalidade implementa validação específica para identificadores únicos (matrícula) e suporta diferentes categorias de combustível. Esta configuração assegura a integridade referencial com o módulo de reservas.

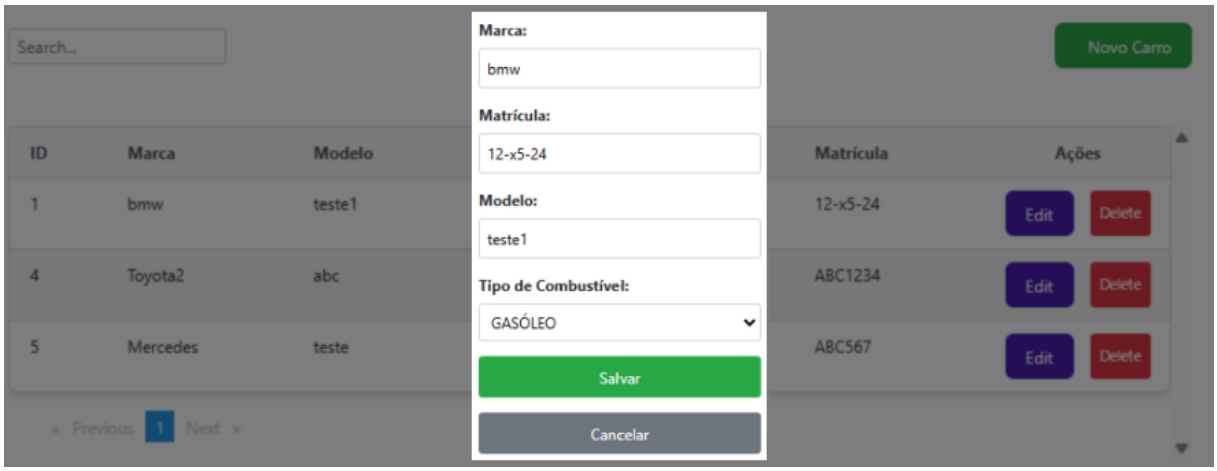


Figura 34 - Gestão de viaturas e formulário de edição

As interfaces administrativas utilizam validação distribuída entre as camadas de apresentação e de lógica de negócio. Esta abordagem assegura consistência operacional e integridade referencial. A aplicação do princípio do menor privilégio é realizada através de *guards* de rota especializados. A arquitetura modular permite extensibilidade futura, com a possibilidade de adição de novos módulos administrativos sem impacto nas funcionalidades existentes.

5.7. Dados Estatísticos/Monitorização *Grafana*

A monitorização operacional implementa *dashboards* analíticos integrados com o frontend Angular através do *Grafana* para análise de padrões de utilização. A implementação combina componentes de interface interativa com sistemas de alerta automático. As visualizações resultam de consultas aplicadas às bases de dados e constituem amostras de teste utilizadas no processo de validação das funcionalidades.

5.7.1. Integração com Frontend

A monitorização e visualização de dados estatísticos do sistema são implementadas através da integração entre o Angular e o *Grafana*, que disponibiliza *dashboards* interativos para análise de métricas operacionais. A integração baseia-se numa arquitetura que incorpora os *dashboards* do *Grafana* diretamente na aplicação web, assegurando a consistência da experiência do utilizador.

O componente `StatisticsGComponent` implementa a camada de integração e utiliza o `DomSanitizer` do Angular, responsável por validar e tratar conteúdos externos antes do seu carregamento através de *iframes*, de forma a prevenir a injeção de código malicioso. A implementação permite o controlo dinâmico dos intervalos temporais através de parâmetros URL, o que possibilita a visualização de dados em períodos pré-definidos (7, 30, 90, 365 dias) ou intervalos personalizados definidos pelo utilizador.

A funcionalidade de controlo temporal é suportada por *reactive forms*, que detetam alterações no intervalo selecionado e atualizam automaticamente o URL do *dashboard*. Esta abordagem permite que os dados apresentados correspondam ao período temporal especificado pelo utilizador, mantendo a sincronização entre a interface e os dados visualizados. A configuração dos *dashboards* utiliza parâmetros específicos do *Grafana* (`kiosk`, `theme=light`) com o objetivo de otimizar a

apresentação no contexto da aplicação, remover elementos desnecessários da interface e adaptar o tema visual à aplicação principal.

5.7.2. Apresentação de Resultados

A Figura 35 apresenta *gauge charts* relativos ao Top 5 de utilizadores com maior número de reservas e maior quilometragem percorrida. Esta representação gráfica foi escolhida por facilitar a comparação visual entre diferentes utilizadores, através de um formato circular com códigos de cor diferenciados.

Figura 35 - Top utilizadores com mais reservas e mais km

A implementação baseia-se em *queries* que agregam dados das tabelas de reservas e utilizadores e calculam as métricas através de operações de contagem e soma. Os resultados revelam disparidades significativas entre a frequência de utilização e a quilometragem efetiva, indicam padrões comportamentais distintos, nomeadamente, utilizadores que privilegiam viagens longas com menor frequência, versus aqueles que efetuam deslocações mais frequentes, mas de menor extensão.

A Figura 36 combina um gráfico de barras dos veículos mais reservados com um gráfico de séries temporais da distância percorrida por dia. O gráfico de barras revela que o BMW lidera com aproximadamente 42 reservas. A visualização temporal complementar mostra a evolução da quilometragem diária, apresentando picos significativos em março e abril de 2025, ultrapassando 250.000 km diários.

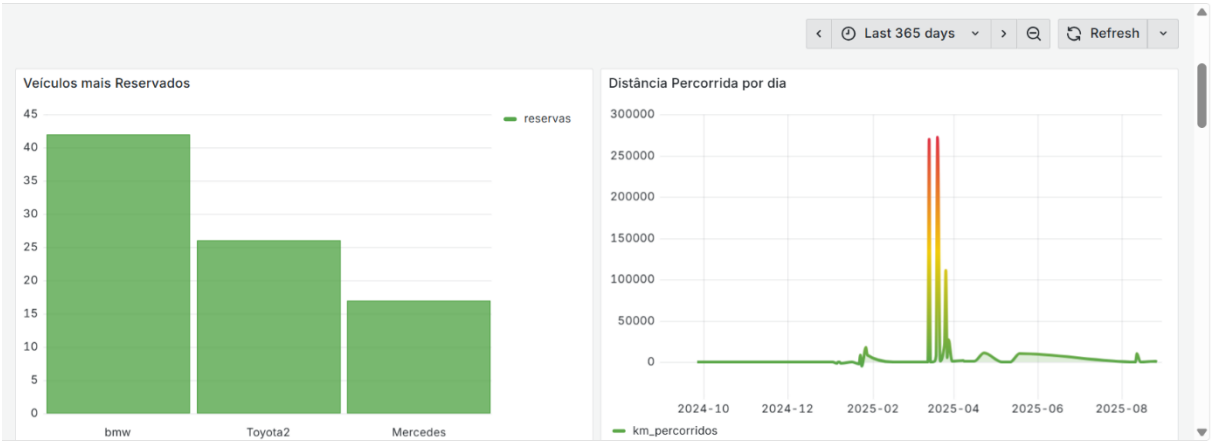


Figura 36 - Veículos mais reservados e maior distâncias por dias

Esta correlação entre frequência de reservas e distância percorrida identifica padrões sazonais de utilização da frota.

A Figura 37, apresenta um *pie chart* e um gráfico de barras para analisar tempos médios de utilização por veículo e distribuição por tipos de autorização. O processamento baseia-se no cálculo de médias de duração entre datas de início e fim das reservas.

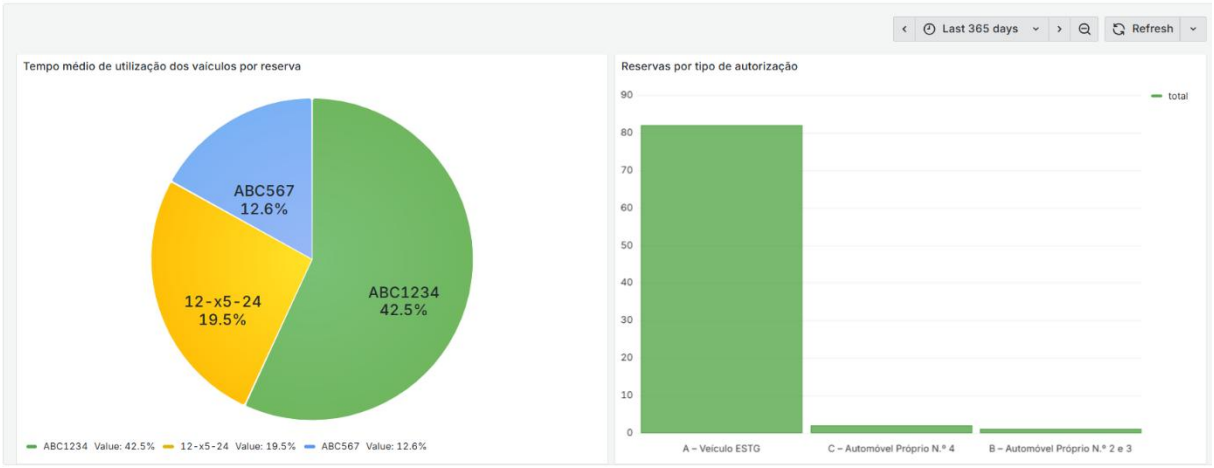


Figura 37 - Média de utilização de veículos por reserva e tipo de autorização

O veículo com a matrícula ABC1234 apresenta 42,5% do tempo total de utilização, o que significa que este veículo tem o maior tempo médio por reserva, não o maior número de reservas. Já a distribuição de autorizações onde o tipo A domina significativamente. Esta disparidade sugere diferentes padrões de aprovação e utilização efetiva.

As visualizações temporais adicionais na Figura 38, através de gráficos de barras horizontais e *bar gauge*, permitem identificar padrões semanais e mensais. As *queries* implementam transformações no formato da data e o agrupamento por períodos específicos.

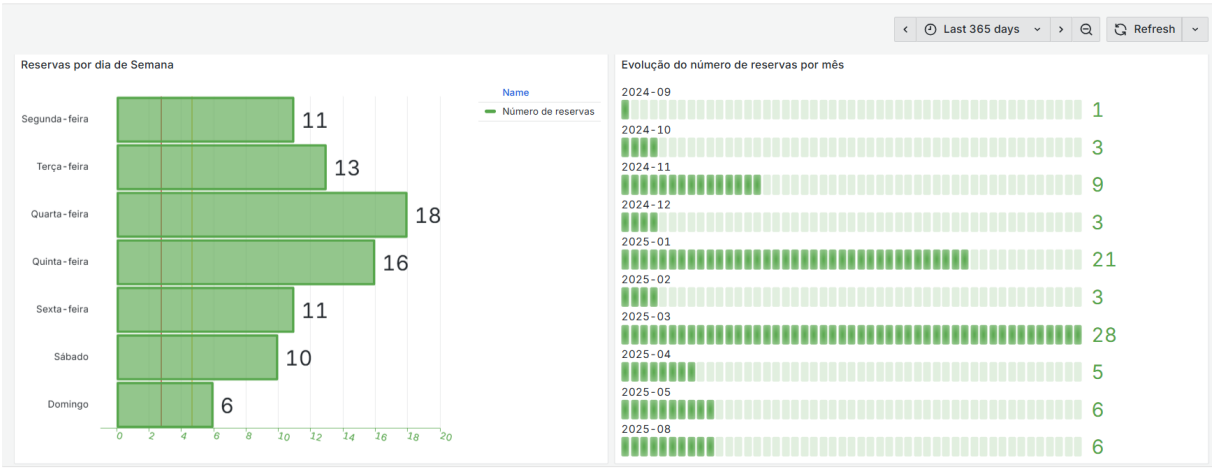


Figura 38 - Reservas por dias de semana e evolução de reservas por mês

A quarta-feira apresenta maior atividade (18 reservas), contrastando com menor utilização ao domingo (6 reservas). O mapa de calor mensal revela picos em janeiro e março, fornecendo dados para otimização da disponibilidade da frota.

A Figura 39 apresenta um mapa geográfico das localizações mais reservadas e um gráfico de barras empilhadas com os horários específicos com maior frequência de reservas. O mapa interativo mostra as localizações através de pontos vermelhos concentrados principalmente na região norte de Portugal.

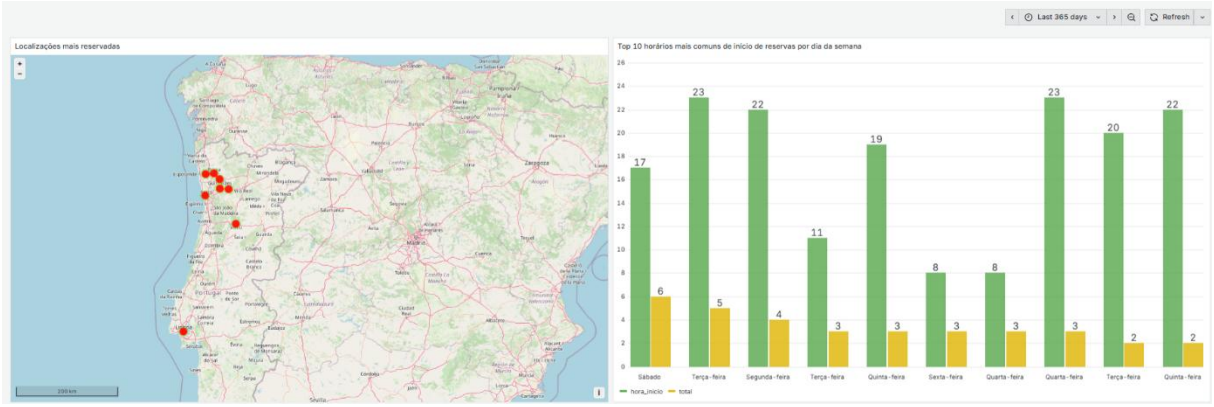


Figura 39 - Localizações e top 10 horários mais comuns de início de reservas por dia da semana

Para os horários mais frequentes, a *query* implementa `DAYNAME()` e `HOUR()` para extrair dia da semana e hora específica, agrupando por ambos os parâmetros. Os resultados mostram que quinta-feira às 14h apresenta o maior pico com 8 reservas, seguido de sábado às 17h (6 reservas). Esta análise granular revela padrões distintos de concentração horária em meio da tarde durante dias úteis e final

da tarde aos fins de semana, permitindo otimizar a disponibilidade da frota nos períodos de maior procura.

A Figura 40 apresenta uma tabela de monitorização de alertas relativa aos tempos de confirmação de realização de reservas. Esta visualização implementa um sistema de alerta automático que identifica utilizadores com tempos de resposta após a data da realização da reserva superiores aos limites estabelecidos. A tabela mostra cinco colunas: UserId, Nome, TempoMedioConfirmacao (dias), TempoMedioKM (dias) e estatuto de Alerta.

UserId	Nome	TempoMedioConfirmacao	TempoMedioKM	Alerta
123	Armando Pereira	47	47	ALERTA
119	Pedro Afonso	1.35	3.42	ALERTA
121	João Manuel Ribeiro	0.409	1.64	
126	Utilizador Testes	0	0	
124	Ana Gomes	-1.50	-1.50	

Figura 40 - Alertas de confirmação de realização

A implementação calcula médias temporais entre o fim da reserva e as respetivas confirmações, classifica automaticamente como "ALERTA" os utilizadores cujas médias excedem 3 dias em qualquer das métricas. Os resultados evidenciam que Armando Pereira apresenta 47 dias em ambas as métricas, enquanto Pedro Afonso e Fernando Martins ultrapassam o limite apenas na indicação de quilómetros.

Esta monitorização integra-se com o microserviço de notificação para envio automático de alertas, permitindo intervenção proactiva em casos de incumprimento dos prazos estabelecidos para confirmação de viagens e registo de quilometragem.

5.8. Testes

A implementação de testes abrangeu quatro níveis distintos: testes unitários do *backend*, testes unitários do *frontend*, validação de *API* e testes funcionais.

5.8.1. Testes Unitários

Os testes unitários do *backend* foram implementados com a *framework* xUnit. A estratégia adotada privilegiou o isolamento de componentes através de *dependency injection* e *mocking*. A implementação seguiu o padrão *Arrange-Act-Assert*, estruturando cada teste em três fases,

nomeadamente, preparação dos dados de entrada, execução da funcionalidade testada e validação dos resultados obtidos. O isolamento de dependências foi realizado através da biblioteca *Moq* que permite a criação de objetos para interfaces como repositórios e serviços de *logging*.

Foram implementados dois tipos principais de testes: testes simples utilizando o atributo `[Fact]` para cenários únicos, e testes parametrizados com `[Theory]` e `[InlineData]` para validar múltiplos cenários com diferentes conjuntos de dados. A execução é realizada através do comando `dotnet test` e obtemos resultados como por exemplo: `Passed! - Failed: 0, Passed: 13, Skipped: 0, Total: 13, Duration: 866 ms - Proj.Tests.dll (net8.0).` Neste exemplo, 13 testes executaram com sucesso, sem falhas ou testes ignorados, e executados em 866 milissegundos.

Os testes unitários do *frontend* foram configurados utilizando Jasmine como *framework* de testes e Karma como *test runner*, integrados nativamente no Angular CLI. A configuração utiliza TestBed para criar módulos de teste isolados, permitindo a configuração de componentes e dependências necessárias. Foi implementado um teste exploratório para validação do AuthService, focando na funcionalidade de autenticação e gestão de *tokens* JWT.

A execução através do comando ng test produz resultados similares, como por exemplo: `Chrome 139.0.0.0 (Windows 10): Executed 1 of 1 SUCCESS (0.015 secs / 0.012 secs).` Neste caso, 1 teste unitário do AuthService foi executado com sucesso no *browser* Chrome, completando em 15 milissegundos.

5.8.2. Validação das API e Testes Funcionais

A validação das API foi realizada através de coleções *Postman* organizadas por microserviço, abrangendo todos os *endpoints* expostos pelos serviços Users, Vehicles, Demo, NotifyFlow, Identity Server e PDF Generator, conforme ilustrado no Apêndice I. A autenticação foi automatizada através de variáveis globais, configuradas para obter e reutilizar tokens *OAuth* 2.0 automaticamente em todas as requisições.

Cada *endpoint* foi testado considerando diferentes cenários: requisições válidas; validação de parâmetros obrigatórios; tratamento de erros e verificação de códigos de resposta HTTP. Os testes incluíram validação de estrutura de dados e confirmação de comportamentos esperados para cada operação CRUD.

Os testes funcionais foram realizados manualmente através do *frontend*, validando o comportamento completo do sistema numa perspetiva *end-to-end*. Esta abordagem permitiu verificar que todas as integrações entre componentes funcionam corretamente no contexto real de utilização.

A validação funcional abrangeu todas as implementações realizadas. Cada funcionalidade foi testada considerando diferentes perfis de utilizador e cenários de utilização típicos, confirmando a correta integração entre todos os componentes da arquitetura implementada.

5.9. Logging

A implementação do sistema de *logging* utiliza o `ILogger` do ASP.NET, configurada centralmente em cada microserviço. A estratégia implementada baseia-se numa abordagem híbrida que combina fornecedores de *logging* distintos para atender aos requisitos de desenvolvimento e produção. Um aspeto da implementação consiste na uniformização do *logging* entre componentes da aplicação e bibliotecas externas, particularmente o *Zeebe* Client, conseguida através da utilização de uma *factory* de *logging* comum. Esta uniformização é necessária para garantir a correlação de eventos em fluxos de trabalho distribuídos, onde a falta de consistência no formato dos *logs* pode comprometer a capacidade de diagnóstico e análise de desempenho. A solução implementada permite que os eventos gerados pelo *Zeebe* sejam processados utilizando o mesmo formato e destino que os eventos da aplicação principal. A implementação categorizando eventos segundo quatro níveis: *Information* para documentar eventos operacionais de rotina; *Warning* para sinalizar situações anómalas que não comprometem o funcionamento; *Error* para falhas que podem afetar funcionalidades; e *Critical* para falhas que tornam o serviço inoperacional, incluindo cenários como perda de conectividade com a base de dados ou indisponibilidade de recursos.

A Figura 41 mostra um exemplo de registo estruturado em diferentes microserviços.

```
01 // Demo API - Evento operacional
02    _logger.LogInformation("Deploy realizado com sucesso: ProcessDefinitionKey
{ProcessDefinitionKey}", _processDefinitionKey);
03
04 // CamundaProxy Controller - Falha não crítica
05 _logger.LogError("Erro ao obter XML,Status: {StatusCode}, Resposta: {Response}",
06     xmlResponse.StatusCode, await xmlResponse.Content.ReadAsStringAsync());
07
08 // Users API - Falha crítica na base de dados
09 _logger.LogCritical(ex, "Critical database failure in Users API - UserId: {UserId} -
service may be unavailable", userId);
```

Figura 41 - Exmplo de logging

Esta implementação proporciona rastreabilidade distribuída através de *logs*, o que facilita o diagnóstico de problemas em ambiente de desenvolvimento, e estabelece a base para integração com sistemas de *logging* distribuído como ELK Stack ou serviços de *cloud logging*, beneficiando da flexibilidade dos *providers* configurados.

5.10. Documentação

A documentação das API é gerada automaticamente através do `Swagger/OpenAPI 3.0`, configurado independentemente em cada microsserviço para fornecer documentação específica por serviço. A configuração utiliza o método `AddSwaggerGen()` com metadados estruturados.

A implementação combina comentários XML do código-fonte com anotações Swagger para gerar documentação interativa. A configuração `IncludeXmlComments()` extrai automaticamente documentação a partir de comentários XML nos controladores e métodos, enquanto `EnableAnnotations()` suporta atributos para descrições dos *endpoints.*

Os *endpoints* são documentados com anotações estruturadas que especificam a funcionalidade, parâmetros de entrada, tipos de resposta e códigos de estado HTTP, conforme exemplificado na Figura 42.

```
01 builder.Services.AddSwaggerGen(c =>
02 {
03     c.SwaggerDoc("v1", new OpenApiInfo {
04         Title = "Vehicles API",
05         Version = "v1",
06         Description = "Microservice for managing vehicles.",
07         Contact = new OpenApiContact { Name = "Backend", Email = "8090457@estg.ipp.pt"
}
08     });
09     var xml = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
10                  c.IncludeXmlComments(Path.Combine(AppContext.BaseDirectory,    xml),
includeControllerXmlComments: true);
11     c.EnableAnnotations();
12     c.ExampleFilters();
13 });
```

Figura 42 - Configuração do Swagger para geração de documentação

A combinação de `[SwaggerOperation]` para descrições funcionais e `[ProducesResponseType]` para especificação de respostas permite documentação importante para quem consume a API.

A estratégia de documentação inclui ainda ficheiros *README* para cada microserviço com instruções de instalação. Esta abordagem híbrida assegura que a documentação técnica das API é mantida automaticamente e sincronizada com o código.

6. Conclusão

Este capítulo apresenta as conclusões do trabalho desenvolvido, sintetiza as principais contribuições técnicas e metodológicas obtidas e identifica as limitações do estudo realizado. São ainda propostas linhas de trabalho futuro que podem estender e aprofundar os resultados alcançados.

6.1. Conclusões e principais contributos

O trabalho realizado validou a viabilidade técnica da integração entre arquiteturas de microserviços e plataformas de gestão de processos de negócio, através de uma implementação prática que demonstra os benefícios desta convergência na transformação digital organizacional.

Do ponto de vista metodológico, foi seguida uma abordagem estruturada para integração entre motores BPM e arquiteturas baseadas em microserviços. Este trabalho documenta desde a identificação de *bounded contexts* até à implementação de comunicação assíncrona, estabelece padrões para projetos similares. A aplicação de *Domain-Driven Design* simplificado revelou-se adequada para orientar a decomposição em microserviços, manteve coesão funcional e reduziu dependências entre componentes. O *Circuit Breaker Pattern* demonstrou capacidade de prevenir falhas em cascata, o API Gateway centralizou o controlo de acesso, enquanto o *Event-Driven Pattern* através do RabbitMQ permitiu o desacoplamento temporal entre serviços. O *Health Check Pattern* estabeleceu monitorização do estado operacional dos componentes, contribuindo para maior observabilidade do sistema distribuído.

A integração com a plataforma Camunda 8 através do Demo API, utilizou o *Zeebe* Client SDK para materializar a coordenação entre *workflows* BPM e microserviços autónomos. Esta abordagem híbrida permite que o motor de processos coordene a execução de *workflows* mantendo a independência dos microserviços.

A estratégia de containerização e orquestração validou diferentes níveis de maturidade tecnológica. A implementação inicial em *Docker* estabeleceu as bases da containerização, enquanto a migração para K3s demonstrou capacidades de balanceamento de carga e distribuição *multi-node*. A adaptação do padrão *Blue-Green Deployment* para ambiente local permitiu reduzir o tempo de indisponibilidade durante atualizações, estabelecendo assim fundamentos para implementações em ambiente de produção.

O sistema de monitorização integrado com *Grafana* transformou dados operacionais em informação útil, implementou *dashboards* analíticos que permitem identificar padrões de utilização e detetar anomalias. O sistema de alertas introduziu capacidades de gestão proativa e substituiu abordagens reativas por monitorização em tempo real.

A validação experimental através da digitalização do processo de autorização de condução de viatura da ESTG demonstrou a aplicabilidade prática da solução proposta. O sistema transformou um procedimento manual num workflow automatizado, evidenciou melhorias em termos de rastreabilidade, consistência operacional e redução de erros humanos.

A integração com a infraestrutura organizacional existente através de protocolos LDAP comprovou que a modernização não exige substituição de sistemas existentes. Esta abordagem híbrida permite transições graduais que preservam investimentos tecnológicos anteriores enquanto introduzem capacidades de gestão de processos.

Os artefactos produzidos, incluindo configurações *Docker*, manifestos K3s, scripts de automação e documentação técnica, constituem contribuições reutilizáveis que podem acelerar implementações similares em contextos organizacionais comparáveis.

6.2. Limitações do estudo e pesquisas futuras

O trabalho realizado apresenta limitações que condicionaram o âmbito dos resultados obtidos e que orientam direções para trabalho futuro.

A integração com o sistema monolítico existente na organização não foi possível concretizar devido à dependência de recursos externos e aprovações organizacionais que extravasaram o período temporal disponível. Esta limitação impediu a validação da interoperabilidade entre a solução desenvolvida e sistemas legados em ambiente operacional real.

A curva de aprendizagem associada às tecnologias adotadas, particularmente a plataforma Camunda, consumiu tempo significativo. A implementação do *frontend* apresenta necessidade de continuidade de desenvolvimento adicional em funcionalidades e interfaces.

As migrações sucessivas de ambiente nativo para *Docker* e posteriormente para K3s revelaram-se desafiantes, exigindo reconfiguração de serviços, resolução de problemas de autenticação e adaptação de padrões de comunicação entre componentes.

A arquitetura de microserviços introduz complexidade operacional que constitui desafio significativo para equipas pequenas. A gestão de todo o ciclo de desenvolvimento, desde o levantamento de requisitos e análise de processos até ao *deployment* e manutenção da solução, exige competências multidisciplinares abrangentes. Os elementos da equipa devem dominar simultaneamente modelação de processos BPMN, arquitetura de microserviços, tecnologias de containerização, configuração de infraestrutura, desenvolvimento *frontend* e *backend*, gestão de bases de dados, padrões de comunicação assíncrona e estratégias de monitorização. Esta dispersão de responsabilidades compromete a profundidade técnica em cada área e aumenta o risco de decisões técnicas inadequadas devido à falta de especialização, especialmente quando comparada com equipas especializadas focadas em domínios específicos da solução.

Trabalhos futuros devem abordar a implementação clássica do padrão *Blue-Green Deployment* em ambiente K3s e explorar capacidades de *load balancing* e *failover* em configurações *multi-node*. Tal permitirá validar a aplicabilidade da solução em contextos de produção com requisitos de alta disponibilidade. A integração com sistemas de logging distribuído como ELK Stack ou serviços de cloud logging constitui uma linha de trabalho que deve ser avaliada futuramente. A aplicação de inteligência artificial à análise de *logs* distribuídos constitui também uma linha de investigação para reduzir a complexidade operacional e melhorar o diagnóstico. A expansão do sistema de monitorização para incluir métricas de negócio complementares às métricas técnicas proporciona visibilidade mais abrangente sobre o impacto organizacional da digitalização de processos.

A extensão da metodologia proposta para outros domínios organizacionais e tipos de processo permitirá validar a generalização da abordagem desenvolvida. A reutilização dos microserviços implementados para digitalização de processos pode acelerar iniciativas de transformação digital, aproveitando os padrões de comunicação, autenticação e monitorização estabelecidos. Esta estratégia de reutilização contribui para a consolidação na integração entre BPM e arquiteturas de microserviços, favorece a maturação desta área de investigação e reduz o esforço de desenvolvimento em projetos subsequentes.

Referências:

[1] C. Matt, T. Hess, and A. Benlian, “Digital Transformation Strategies,” *Business and Information Systems Engineering*, vol. 57, no. 5, pp. 339–343, Oct. 2015, doi: 10.1007/s12599-015-0401-5.

[2] Y. Liu, Z. Ni, M. Karlsson, and S. Gong, “Methodology for digital transformation with internet of things and cloud computing: A practical guideline for innovation in small-and medium-sized *Enterprise*s,” *Sensors*, vol. 21, no. 16, Aug. 2021, doi: 10.3390/s21165355.

[3] M. S. Denner, L. C. Püschel, and M. Röglinger, “How to Exploit the Digitalization Potential of Business Processes,” *Business and Information Systems Engineering*, vol. 60, no. 4, pp. 331–349, Aug. 2018, doi: 10.1007/s12599-017-0509-x.

[4] M. Dumcius and T. Skersys, “Improvement and digitalization of business processes in small-medium *Enterprise*s,” 2019, Accessed: Nov. 16, 2023. [Online]. Available: https://epubl.ktu.edu/object/elaba:42610837/

[5] N. Kovalevska, I. Nesterenko, O. Lutsenko, O. Nesterenko, and Y. Hlushach, “Problems of accounting digitalization in conditions of business processes digitalization,” *Revista Amazonia Investiga*, vol. 11, no. 56, pp. 132–141, Oct. 2022, doi: 10.34069/ai/2022.56.08.14.

[6] J. Myslin and J. Kaiser, “State Modeling Methodology for Business Processes,” *TEM Journal*, vol. 11, no. 4, pp. 1824–1834, Nov. 2022, doi: 10.18421/TEM114-50.

[7] J. Jeston and J. Nelis, *Business Process Management: Practical Guidelines to Successful Implementations*. London, U.K.: Routledge, 2014. doi: 10.4324/9780203081327.

[8] A. Ciaramella, M. G. C. A. Cimino, B. Lazzerini, and F. Marcelloni, “Using BPMN and tracing for rapid business process prototyping environments,” *ICEIS 2009 - 11th International Conference on Enterprise Information Systems, Proceedings*, vol. ISAS, pp. 206–212, 2009, doi: 10.5220/0002005002060212.

[9] L. Zineb, R. Maryam, S. Rajaa, and R. Moulay, “A set of indicators for BPM life cycle improvement,” *IEEE*, 2018, doi: 10.1109/ISACV.2018.8354057.

[10] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*. Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-33143-5.

[11] Object Management Group, “Business Process Model and Notation (BPMN),” Dec. 2013. Accessed: Jun. 22, 2025. [Online]. Available: https://www.omg.org/spec/BPMN/2.0.2/PDF

[12] L. Mendoza Morales, “Business Process Veri cation: The Application of Model Checking and Timed Automata,” *CLEI Electronic Journal*, vol. 17, pp. 1080–1095, Jun. 2014, doi: 10.19153/cleiej.17.2.2.

[13] P. Valderas, V. Torres, and E. Serral, “Modelling and executing IoT-enhanced business processes through BPMN and microservices,” *Journal of Systems and Software*, vol. 184, p. 111139, 2022, doi: 10.1016/j.jss.2021.111139.

[14] D. Lübke, M. Ahrens, and K. Schneider, “Influence of diagram layout and scrolling on understandability of BPMN processes: an eye tracking experiment with BPMN diagrams,” *Information Technology and Management*, vol. 22, no. 2, pp. 99–131, 2021, doi: 10.1007/s10799-021-00327-7.

[15] M. Chebrolu, “Building Scalable Applications: Transitioning from Monolithic to Microservices with Modularized Design,” *International Research Journal of Modernization in Engineering Technology and Science*, vol. 7, no. 3, pp. 1–8, 2025.

[16] E. Taibi, D. Lenarduzzi, and C. Pahl, “Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation,” *IEEE Cloud Computing*, vol. 4, no. 5, pp. 22–32, Oct. 2017, doi: 10.1109/MCC.2017.4250931.

[17] G. Blinowski, A. Ojdowska, and A. Przybyłek, “Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation,” *IEEE Access*, vol. 10, pp. 20357–20374, 2022, doi: 10.1109/ACCESS.2022.3152803.

[18] K. Barde, “Modular Monoliths: Revolutionizing Software Architecture for Efficient Payment Systems in Fintech,” *International Journal of Computer Trends and Technology*, vol. 71, pp. 20–27, 2023, doi: 10.14445/22312803/IJCTT-V71I10P103.

[19] N. Niknejad, W. Ismail, I. Ghani, B. Nazari, M. Bahari, and A. B. Hussin, “Understanding Service-Oriented Architecture (SOA): A systematic literature review and directions for further investigation,” *Inf Syst*, vol. 91, p. 101491, 2020, doi: 10.1016/j.is.2020.101491.

[20] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall PTR, 2004.

[21] E. Newcomer and G. Lomow, *Understanding SOA with Web Services*. Addison-Wesley, 2005.

[22] B. Manouvrier and L. Menard, *Application integration : EAI, B2B, BPM and SOA*. ISTE, 2008.

[23] S. Pallewatta, V. Kostakos, and R. Buyya, “Placement of Microservices-based IoT Applications in Fog Computing: A Taxonomy and Future Directions,” 2022.

[24] R. Buyya and others, “A Manifesto for Future Generation Cloud Computing: Research Directions for the Next Decade,” *ACM Comput Surv*, vol. 51, no. 5, pp. 1–38, 2018, doi: 10.1145/3241737.

[25] S. K. Singh, P. S. Srinivasan, and D. Kaur, “SOA Cloud Computing: Modernized the Supply Chain Management Applications,” *IOSR Journal of Engineering*, vol. 9, no. 3, pp. 67–74, Dec. 2024.

[26] B. and B. J. J. and A. A. Costa, “Orchestration in Fog Computing: A Comprehensive Survey,” *ACM Comput Surv*, vol. 55, pp. 01–34, Jan. 2022, doi: 10.1145/3486221.

[27] D. Guinard and V. Trifa, *Building the Web of Things: With examples in Node.js and Raspberry Pi*. Manning Publications, 2016.

[28] Y. Lu, “Industry 4.0: A survey on technologies, applications and open research issues,” *J Ind Inf Integr*, vol. 6, pp. 1–10, 2017, doi: 10.1016/j.jii.2017.04.005.

[29] T. Torres, G. Gonçalves, and R. Pinto, “Literature Review on Cloud-Based Service-Oriented Architecture for IEC 61499 Distributed Control Systems,” in *Proceedings of the 15th International Conference on Cloud Computing and Services Science (CLOSER 2025)*, 2025, pp. 174–181. doi: 10.5220/0013293400003950.

[30] S. Zhang, “Apply SOA Paradigms in Cyber-Physical System to Enhance Interoperability: State-of-the-Art Review,” Mar. 2019. [Online]. Available: https://arxiv.org/abs/1903.00065

[31] A. Nadalin, C. Kaler, P. Hallam-Baker, and R. Monzillo, “Web Services Security: SOAP Message Security 1.1 (WS-Security 2004),” Feb. 2006. [Online]. Available: https://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[32] Z. Maamar, N. Faci, and H. Yahyaoui, “Security in SOA: A Survey,” *Comput Secur*, vol. 105, p. 102237, Mar. 2021, doi: 10.1016/j.cose.2021.102237.

[33] Kat *et al.*, “Hybrid Cloud Connector: Offloading integration complexities,” Sep. 2024, pp. 196–197. doi: 10.1145/3688351.3689168.

[34] S. Newman, *Building Microservices*, 2nd ed. O’Reilly Media, 2021.

[35] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Boston: Addison-Wesley, 2012.

[36] C. Richardson, *Microservices Patterns: With Examples in Java*. Shelter Island, NY, USA: Manning Publications, 2019.

[37] N. Dragoni and et al., “Microservices: Yesterday, Today, and Tomorrow,” in *Present and Ulterior Software Engineering*, Springer, 2017, pp. 195–216. doi: 10.1007/978-3-319-67425-4_12.

[38] A. Ganje, “Microservices in Organizations,” *Journal of Software Engineering and Applications*, vol. 18, no. 2, pp. 76–86, 2025, doi: 10.4236/jsea.2025.182005.

[39] V. Velepucha and P. Flores, “A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges,” *IEEE Access*, vol. 11, pp. 88339–88358, 2023, doi: 10.1109/ACCESS.2023.3305687.

[40] M. Kleppmann, *Designing Data-Intensive Applications*. O’Reilly Media, 2017.

[41] M. Waseem and others, “Design, Monitoring, and Testing of Microservices Systems: The Practitioners’ Perspective,” 2021. doi: 10.1016/j.jss.2021.111061.

[42] J. Willard and J. Hutson, “The Evolution and Future of Microservices Architecture with AI-Driven Enhancements,” *International Journal of Recent Engineering Science*, vol. 12, no. 1, pp. 16–22, 2025, doi: 10.14445/23497157/IJRES-V12I1P103.

[43] R. Su, X. Li, and D. Taibi, “From Microservice to Monolith: A Multivocal Literature Review,” *Electronics (Basel)*, vol. 13, no. 8, p. 1452, 2024, [Online]. Available: https://www.mdpi.com/2079-9292/13/8/1452

[44] M. K. Dehnoi, “Harnessing the Power of Service-Oriented Organization Architecture (SOA) for Business Transformation,” 2025. [Online]. Available: https://www.researchgate.net/publication/389661158_Harnessing_the_Power_of_Service-Oriented_Organization_Architecture_SOA_for_Business_Transformation

[45] L. Chamari, E. Petrova, and P. Pauwels, “An End-to-End Implementation of a Service-Oriented Architecture for Data-Driven Smart Buildings,” *IEEE Access*, vol. 11, pp. 117261–117281, 2023, doi: 10.1109/ACCESS.2023.3325767.

[46] M. Seedat, Q. Abbas, and N. Ahmad, “Systematic Mapping of Monolithic Applications to Microservices Architecture,” 2023. [Online]. Available: https://arxiv.org/abs/2309.03796

[47] V. L. Nogueira, F. S. Felizardo, A. M. M. Amaral, W. K. G. Assunção, and T. E. Colanzi, “Insights on Microservice Architecture Through the Eyes of Industry Practitioners,” 2024. [Online]. Available: https://arxiv.org/abs/2408.10434

[48] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect’s Perspective*, 1st ed. Boston, MA: Addison-Wesley, 2015.

[49] Camunda, “Deutsche Telekom modernizes their process automation architecture and maximizes their RPA investment,” Jul. 2020. Accessed: Jun. 28, 2025. [Online]. Available: https://camunda.com/case-study/deutsche-telekom/

[50] Bizagi, “Leading Colombian bank re-engineers systems with low-code automation for customer-centricity,” 2024. Accessed: Jun. 30, 2025. [Online]. Available: https://go.bizagi.com/rs/741-TEJ-653/images/Banco_de_Occidente_Bizagi_Case_Study.pdf

[51] A. Đurić, A. Hornjak, D. Stefanović, and D. Dakić, “Enhancing Production Planning Efficiency with Document Management Using BPM Tools: The Bonita Approach,” in *Proceedings of the International May Conference on Strategic Management (IMCSM)*, Bor, Serbia, May 2024, pp. 282–291. doi: 10.5937/IMCSM24028D.

[52] Bonitasoft, “Bonita Documentation – Architecture and Features.” Accessed: Jul. 01, 2025. [Online]. Available: https://documentation.bonitasoft.com/

[53] Camunda, “Camunda 8 Documentation,” 2025. Accessed: Jun. 10, 2025. [Online]. Available: https://docs.camunda.io/

[54] D. Meyer, “Camunda Licensing: What You Need to Know 8.6,” Apr. 08, 2024. Accessed: Jul. 10, 2025. [Online]. Available: https://camunda.com/blog/2024/10/camunda-licensing-what-you-need-to-know/

[55] Bizagi, “Bizagi Documentation,” 2025. Accessed: Jul. 06, 2025. [Online]. Available: https://help.bizagi.com/platform/en/index.html

[56] PeerSpot, “Bizagi: Reviews, Tips and Advice from Real Users,” 2025. Accessed: Jul. 05, 2025. [Online]. Available: https://www.peerspot.com/products/bizagi-reviews

[57] Bonitasoft, “Bonita BPM platform overview,” 2025. Accessed: Jun. 01, 2025. [Online]. Available: https://documentation.bonitasoft.com/bonita/latest/bonita-overview/bonita-bpm-overview

[58] PeerSpot, “Bonita – Reviews, Tips and Advice from Real Users July 2025,” 2025. Accessed: Jul. 01, 2025. [Online]. Available: https://www.peerspot.com/products/bonita-pros-and-cons

[59] PeerSpot Community, “Camunda: Reviews, Tips and Advice from Real Users,” 2025. Accessed: May 10, 2025. [Online]. Available: https://www.peerspot.com/products/camunda-reviews

[60] O. Özkan, Ö. Babur, and M. van den Brand, “*Domain-Driven Design* in Software Development: A Systematic Literature Review on Implementation, Challenges, and Effectiveness,” *Journal of Systems and Software*, vol. 230, Jun. 2025, doi: 10.1016/j.jss.2025.112537.

[61] Microsoft, “Implementing the Circuit Breaker pattern,” 2023. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/implement-resilient-applications/implement-circuit-breaker-pattern

[62] Microsoft, “The API gateway pattern versus the direct client-to-microservice communication - .NET.” Accessed: Jul. 15, 2025. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern

[63] B. Yang, A. Sailer, S. Jain, A. E. Tomala-Reyes, M. Singh, and A. Ramnath, “Service Discovery Based *Blue-Green Deployment* Technique in Cloud Native Environments,” in *2018 IEEE International Conference on Services Computing (SCC)*, Jul. 2018, pp. 185–192. doi: 10.1109/SCC.2018.00031.

[64] Microsoft, “ASP.NET Core.” Accessed: Feb. 25, 2025. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/

[65] Microsoft, “Entity *Framework* Core.” Accessed: Jan. 20, 2025. [Online]. Available: https://learn.microsoft.com/en-us/ef/core/

[66] Duende Software, “Duende IdentityServer Documentation.” Accessed: Jan. 25, 2025. [Online]. Available: https://docs.duendesoftware.com/identityserver/v6/

[67] Swashbuckle, “Swashbuckle.AspNetCore.” Accessed: Mar. 02, 2025. [Online]. Available: https://github.com/domaindrivendev/Swashbuckle.AspNetCore

[68] RabbitMQ, “Messaging that just works.” Accessed: Apr. 12, 2025. [Online]. Available: https://www.rabbitmq.com/

[69] ThreeMammals, “Ocelot API Gateway Documentation.” Accessed: Mar. 22, 2025. [Online]. Available: https://ocelot.readthedocs.io/

[70] Angular Team, “What is Angular?” Accessed: Feb. 05, 2025. [Online]. Available: https://angular.io/guide/what-is-angular

[71] *Docker*, “*Docker* Overview.” Accessed: Feb. 05, 2025. [Online]. Available: https://docs.*Docker*.com/get-started/overview/

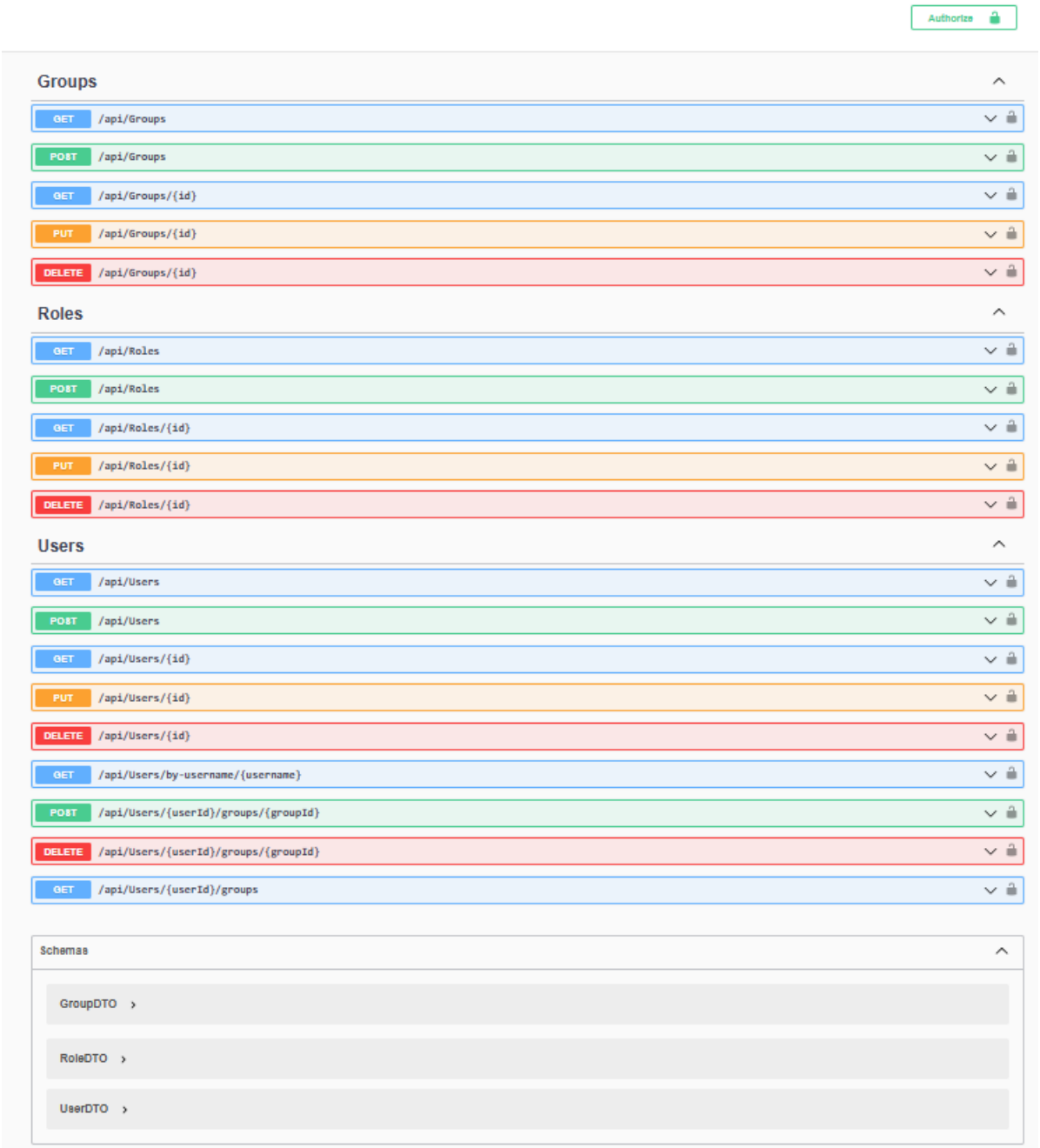
[72] Portainer.io, “Portainer: Simplifying Container Management.” Accessed: Feb. 06, 2025. [Online]. Available: https://www.portainer.io/

[73] H. Koziolek and N. Eskandani, “Lightweight *Kubernetes* Distributions: A Performance Comparison of MicroK8s, k3s, k0s, and Microshift,” in *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE ’23)*, Coimbra, Portugal: ACM, 2023, pp. 1–13. doi: 10.1145/3578244.3583737.

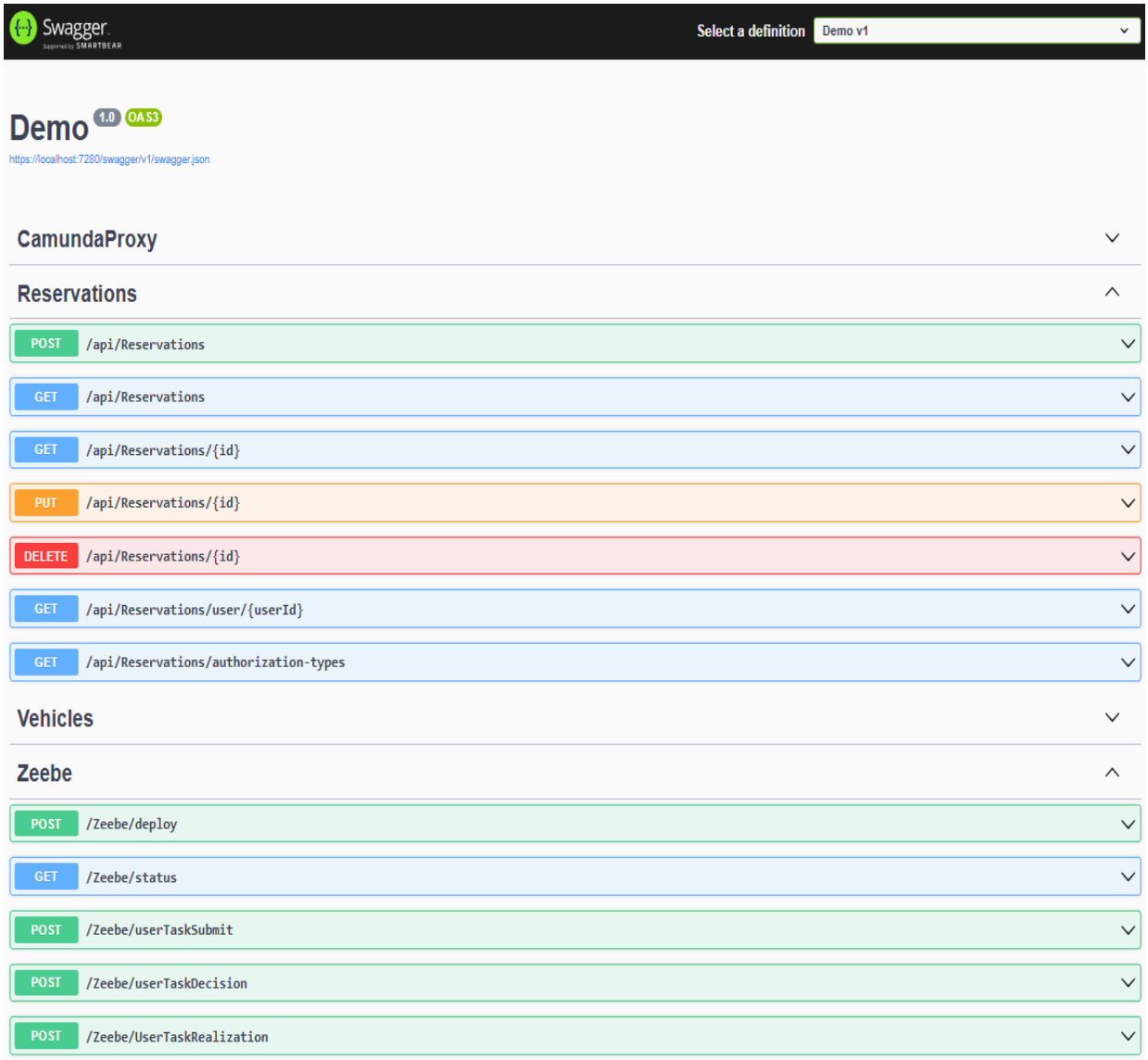
[74] *Grafana* Labs, “*Grafana* documentation.” Accessed: Dec. 01, 2024. [Online]. Available: https://*Grafana*.com/docs/*Grafana*/latest/introduction/

Apêndices

Apêndice A Endpoints Users API



Apêndice B Endpoints do Demo API



Apêndice C Exemplo PDF gerado

P.PORTO

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Requisição de Viatura

Número de Pedido: 103

Decisão/Despacho

Decisão: Deferido **Por:**Autoriza Testes

Data: 26/08/2025

Informações: Concordo

Por delegação nos termos do Despacho n.º 1294/2023 (D.R., 2.ª série, n.º 18, de 25 de janeiro).

(1) **Utilizador Testes**, (2) **General**, da Escola Superior de Tecnologia e Gestão do Politécnico do Porto, solicita autorização para:

A Conduzir o veículo de serviços gerais da ESTG Marca: **bmw** , Modelo: **teste1**, Matrícula: **12-x5-24**.

Para efetuar o seguinte serviço da Escola: **Formação Porto**

Percurso: Felgueiras - Porto - Felgueiras

Início: 27/08/2025 08:00 **Regresso previsto:** 27/08/2025 20:00

Data do Pedido: 26/08/2025

A preencher pelo funcionário (apenas para veículos da Escola):

Quilómetros à partida: 20500

Quilómetros à chegada: 21000

Ocorrências/Anomalias detectadas no veículo: s/a

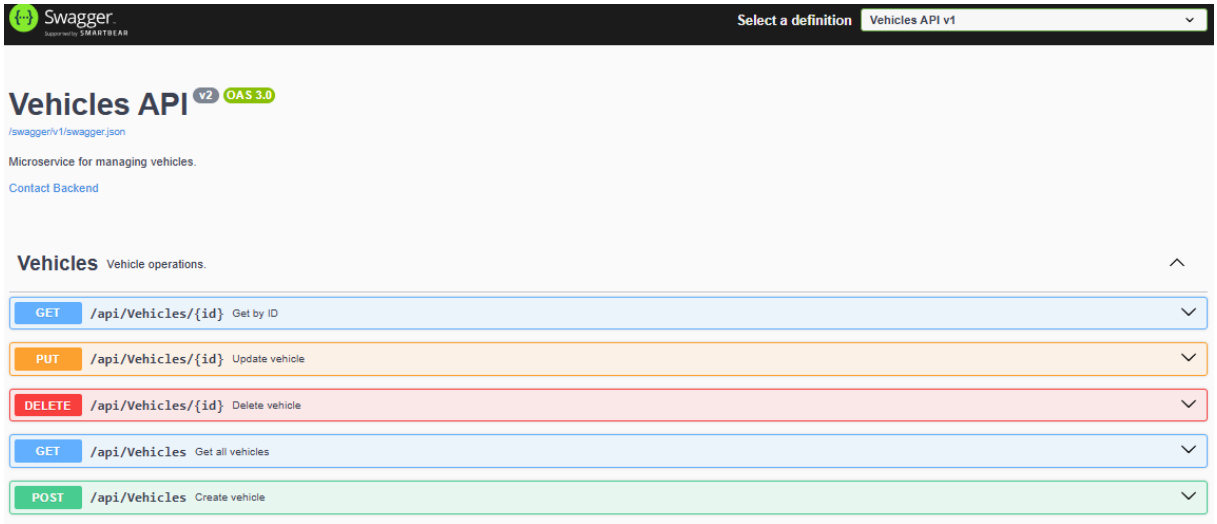
---

(1) Nome do funcionário

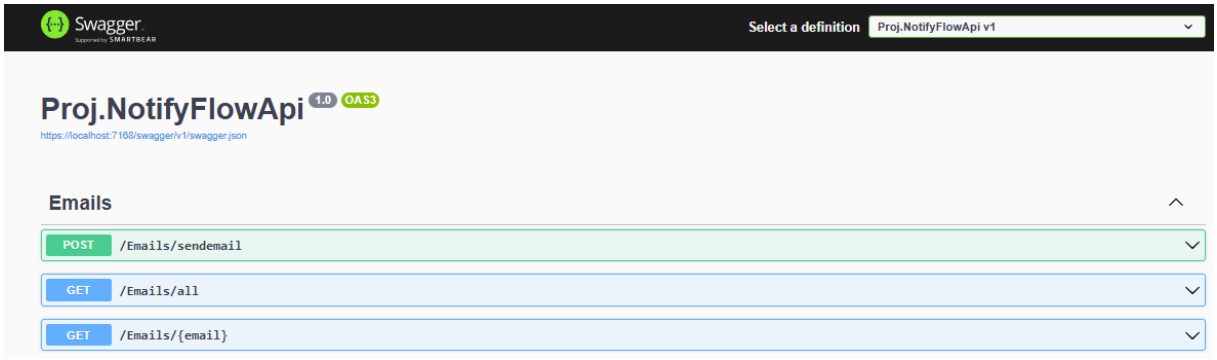
(2) Categoria

Documento processado eletronicamente | Código de segurança: {{SecurityCode}}

Apêndice D Endpoints Vehicles API



Apêndice E Endpoints NotifyFlow API



Apêndice F Deployment.yaml K3s

```
01  apiVersion: apps/v1
02  kind: Deployment
03  metadata:
04    name: pdf-generator-deployment
05    labels:
06      app: pdf-generator
07  spec:
08    replicas: 1
09    selector:
10      matchLabels:
11        app: pdf-generator
12    template:
13      metadata:
14        labels:
15          app: pdf-generator
16      spec:
17        containers:
18          - name: pdf-generator
19            image: pdf-generator-service:latest
20            imagePullPolicy: Never
```

```
21            ports:
22              - containerPort: 8080
23            env:
24              - name: ASPNETCORE_ENVIRONMENT
25                value: "Development"
26              - name: ASPNETCORE_URLS
27                value: "https://+:8080"
28              - name: Kestrel__Endpoints__Https__Certificate__Path
29                value: "/certs/aspnetapp.pfx"
30              - name: Kestrel__Endpoints__Https__Certificate__Password
31                value: "***"
32            volumeMounts:
33              - name: generated-pdfs
34                mountPath: /app/GeneratedPdfs
35              - name: certs
36                mountPath: /certs
37        volumes:
38          - name: generated-pdfs
39            hostPath:
40              path: /mnt/microservicos/PdfGeneratorService/GeneratedPdfs
41          - name: certs
42            hostPath:
43              path: /mnt/microservicos/certs
```

Apêndice G Service.yaml k3s

```
01  apiVersion: v1
02  kind: Service
03  metadata:
04    name: pdf-generator-service
05    labels:
06      app: pdf-generator
07    annotations:
08 traefik.ingress.Kubernetes.io/service.serversscheme: https
09 traefik.ingress.Kubernetes.io/service.serverstransport:insecure-skip-verify@Kubernetescrd
10  spec:
11    selector:
12      app: pdf-generator
13    ports:
14      - name: https
15        protocol: TCP
16        port: 8080
17        targetPort: 8080
```

Apêndice H Script de Teste Load Balancing

```
01  # Kubernetes Load Balancing Test Script - Pedro Manuel Pereira Afonso
02  if (-not ([Sys-
tem.Management.Automation.PSTypeName]'TrustAllCertsPolicy').Type) {
03      add-type @"
04  using System.Net; using System.Security.Cryptography.X509Certificates;
05  public class TrustAllCertsPolicy : ICertificatePolicy {
06      public bool CheckValidationResult(ServicePoint srvPoint, X509Certificate c
ertificate, WebRequest request, int certificateProblem) { return true; }
07  }
08  "@
```

```
09  }
10  [System.Net.ServicePointManager]::CertificatePolicy = New-Object TrustAll-
CertsPolicy
11  $apiUrl = "https://localhost:7281/api/generatepdf"
12  function Get-PodInfo {
13      $pods = kubectl get pods -l app=pdf-generator -o wide --no-headers
14      $activePods = @()
15      $pods | ForEach-Object {
16          $parts = $_ -split '\s+'
17          if ($parts[2] -eq "Running") {
18              $activePods += @{ Name = $parts[0]; IP = $parts[5]; Node = $parts[
6] }
19              Write-
Host "Pod: $($parts[0]) | IP: $($parts[5]) | Node: $($parts[6])"
20          }
21      }
22      Write-Host "Active pods: $($activePods.Count)"
23      return $activePods
24  }
25  function Get-CPUData {
26      $metrics = @{}
27      kubectl top pods -l app=pdf-generator --no-headers | ForEach-Object {
28          $parts = $_ -split '\s+'
29          if ($parts.Length -ge 2) { $metrics[$parts[0]] = $parts[1] }
30      }
31      return $metrics
32  }
33  function Test-APIConnection {
34      try {
35          Invoke-RestMethod -Uri $apiUrl -Method Post -Body '{"id":0}' -
ContentType "application/json" -TimeoutSec 5 | Out-Null
36          return $true
37      } catch { return $false }
38  }
39  function Run-LoadTest {
40      param($testName, $requestCount)
41      Write-Host "`n--- $testName ---"
42      if (-not (Test-APIConnection)) {
43          Write-Host "API ERROR: Check port-forward"
44          return $null
45      }
46      $pods = Get-PodInfo
47      $initialCPU = Get-CPUData
48      Write-Host "Initial CPU:" -NoNewline
49      foreach ($pod in $pods) {
50          if ($initialCPU.ContainsKey($pod.Name)) {
51              Write-Host " $($pod.Name):$($initialCPU[$pod.Name])" -NoNewline
52          }
53      }
54      Write-Host ""
55      $testStart = Get-Date
56      $requestTimes = @()
57      $successCount = 0
58      for ($i = 1; $i -le $requestCount; $i++) {
59          $body = @{ id = (12000 + $i) } | ConvertTo-Json
60          $reqStart = Get-Date
61          try {
```

```
62              Invoke-RestMethod -Uri $apiUrl -Method Post -Body $body -
ContentType "application/json" -TimeoutSec 10 | Out-Null
63              $requestTimes += ((Get-Date) - $reqStart).TotalMilliseconds
64              $successCount++
65              if ($i % 5 -eq 0) { Write-Host "Progress: $i/$requestCount" }
66          } catch { Write-Host "Request $i failed" }
67          Start-Sleep -Milliseconds 250
68      }
69      $totalTime = ((Get-Date) - $testStart).TotalSeconds
70      Start-Sleep -Seconds 2
71      $finalCPU = Get-CPUData
72      Write-Host "Final CPU:" -NoNewline
73      foreach ($pod in $pods) {
74          if ($finalCPU.ContainsKey($pod.Name)) {
75              Write-Host " $($pod.Name):$($finalCPU[$pod.Name])" -NoNewline
76          }
77      }
78      Write-Host ""
79      if ($requestTimes.Count -gt 0) {
80          $avgTime = ($requestTimes | Measure-Object -Average).Average
81          $minTime = ($requestTimes | Measure-Object -Minimum).Minimum
82          $maxTime = ($requestTimes | Measure-Object -Maximum).Maximum
83          $throughput = $successCount / $totalTime
84          Write-
Host "RESULTS: Success $successCount/$requestCount | Avg ${avgTime}ms | Min ${minT
ime}ms | Max ${maxTime}ms | throughput ${throughput} req/s"
85          return @{
86              PodCount = $pods.Count
87              SuccessCount = $successCount
88              TotalTime = $totalTime
89              AvgTime = $avgTime
90              MinTime = $minTime
91              MaxTime = $maxTime
92              throughput = $throughput
93              SuccessRate = ($successCount / $requestCount) * 100
94          }
95      }
96      return $null
97  }
98  Write-Host "KUBERNETES LOAD BALANCING TEST - $(Get-Date -Format 'HH:mm:ss')"
99  $result1 = Run-LoadTest "MULTI-POD TEST" 20
100 if (-not $result1) { Write-Host "Phase 1 failed"; exit 1 }
101 $currentReplicas = kubectl get deployment pdf-generator-deployment -o json-
path='{.spec.replicas}'
102 Write-Host "`nScaling $currentReplicas -> 1 replica"
103 kubectl scale deployment pdf-generator-deployment --replicas=1
104 for ($i = 10; $i -gt 0; $i--) { Write-Host "Waiting ${i}s"; Start-Sleep -
Seconds 1 }
105 $result2 = Run-LoadTest "SINGLE POD TEST" 20
106 Write-Host "`nRestoring to $currentReplicas replicas"
107 kubectl scale deployment pdf-generator-deployment --replicas=$currentReplicas
108 Start-Sleep -Seconds 5
109 if ($result1 -and $result2) {
110     Write-Host "`n--- ANALYSIS ---"
111     Write-Host "Multi-
pod: $($result1.PodCount) pods | $([math]::Round($result1.AvgTime))ms avg | $([mat
h]::Round($result1.Throughput, 2)) req/s"
```

```
112     Write-Host "Single-
pod: $($result2.PodCount) pod | $([math]::Round($result2.AvgTime))ms avg | $([math
]::Round($result2.Throughput, 2)) req/s"
113     Write-
Host "Performance difference: $([math]::Round($result2.AvgTime - $result1.AvgTime)
)ms"
114     Write-
Host "VALIDATED: Load balancing, horizontal scaling, resilience, multi-
node distribution"
115 }
116 Write-Host "Test completed: $(Get-Date -Format 'HH:mm:ss')"
```

Script PowerShell que implementa testes de *load balancing* e escalonamento horizontal em clusters K3D. Executa duas fases de teste (multi-pod e single-pod) com 20 pedidos cada, monitoriza métricas de CPU e tempos de resposta, demonstrando distribuição de carga, resiliência.

Apêndice I Exemplo Coleções *Postman*