Integração e Otimização do Onlinedesk através do eProcess para gestão de processos End-to-End

Mestrado em Engenharia Informática – Computação Móvel

Bernardo José Lopes Ferreira

Leiria, julho de 2025

Integração e Otimização do Onlinedesk através do eProcess para gestão de processos End-to-End

Mestrado em Engenharia Informática – Computação Móvel

Bernardo José Lopes Ferreira

Projeto realizado sob a orientação do Professor Marco António de Oliveira Monteiro e sob supervisão do Miguel Santos Silva Baptista de Almeida

Leiria, julho de 2025

Originalidade e Direitos de Autor

O presente relatório de projeto é original, elaborado unicamente para este fim, tendo sido devidamente citados todos os autores cujos estudos e publicações contribuíram para o elaborar.

Reproduções parciais deste documento serão autorizadas na condição de que seja mencionado o autor e feita referência ao ciclo de estudos no âmbito do qual o mesmo foi realizado, a saber, Mestrado em Engenharia Informática – Computação Móvel, no ano letivo 2024/2025 da Escola Superior de Tecnologia e Gestão (ESTG) do Instituto Politécnico de Leiria (IPL), Portugal, e, bem assim, à data das provas públicas que visaram a avaliação destes trabalhos (se aplicável).

Dedicatória

À minha família, em especial aos meus pais, irmãs e namorada, pelo apoio constante, carinho, compreensão e encorajamento ao longo de todo este percurso e nos momentos de maior desafio. Aos amigos que, com a sua presença e incentivo, tornaram esta etapa mais leve e possível.

Agradecimentos

Gostaria de expressar a minha sincera gratidão ao Professor Marco António de Oliveira Monteiro, pela orientação, incentivo, acompanhamento atento, sugestões valiosas e disponibilidade ao longo do desenvolvimento desta dissertação. A sua experiência e apoio foram fundamentais para definir a direção e qualidade deste trabalho.

Agradeço também profundamente ao meu orientador e supervisor deste projeto na empresa, que também é meu líder técnico e responsável técnico pelo componente OnlineDesk, Miguel Santos Silva Baptista de Almeida, cujo apoio, orientação e conhecimento técnico foram determinantes para o sucesso do projeto. O seu acompanhamento constante e visão estratégica foram fundamentais para ultrapassar os principais desafios encontrados.

Estendo também os meus agradecimentos ao Fernando Jorge Costa da Silva Faria, Diretor da minha unidade, pelo voto de confiança e apoio, bem como ao Filipe Mendes Correia, líder técnico do Edoclink, pela orientação e colaboração contínuas ao longo do desenvolvimento.

Agradeço também a todos os colegas da empresa que, de diferentes formas, contribuíram para o sucesso deste projeto. A sua disponibilidade, feedback e colaboração foram essenciais durante a implementação da minha parte do desenvolvimento da framework eProcess.

Declaração sobre a Utilização de Inteligência Artificial

Gostaria de reconhecer a utilização do ChatGPT (https://openai.com/index/chatgpt/), desenvolvido pela OpenAI, como ferramenta de apoio durante o processo de redação deste documento. O modelo foi usado para auxiliar na melhoria da coesão e clareza textual, para garantir uniformidade e na revisão linguística. Todo o conteúdo gerado foi cuidadosamente revisto, adaptado e validado pelo autor, sendo a responsabilidade final da redação, decisões técnicas e científicas inteiramente do mesmo.

Resumo

Esta tese apresenta o desenvolvimento e integração de várias melhorias arquiteturais no OnlineDesk, um componente da framework eProcess, uma plataforma empresarial desenvolvida pela Link Consulting para apoiar a conceção, execução e monitorização de processos de negócio. A framework integra componentes modulares para gestão de formulários, ciclo de vida documental e monitorização de processos, oferecendo uma solução coesa para BPM (do inglês *Business Process Management*). Neste contexto, o OnlineDesk facilita a recolha estruturada de dados, a submissão de pedidos com base em formulários e a interação com outros sistemas *backend*. O trabalho foi realizado no âmbito de um projeto real de uma empresa, com o objetivo de melhorar a interoperabilidade, a manutenibilidade e a extensibilidade, através da refatorização do protocolo de comunicação do OnlineDesk com o Edoclink, substituindo SOAP (do inglês *Simple Object Access Protocol*) por APIs (do inglês *Application Programming Interfaces*) REST (do inglês *Representational State Transfer*), da integração de suporte a formulários dinâmicos com o Form Builder, da introdução de monitorização de ciclo de vida baseado em eventos via Kafka e Process Monitor, e da implementação de um mecanismo de importação baseado em JSON (do inglês *JavaScript Object Notation*) para permitir a portabilidade de configurações de formulários e serviços.

Estas melhorias foram desenvolvidas de forma iterativa e validadas através de testes contínuos e feedback das partes interessadas. Posteriormente, foram integradas em cenários piloto reais e demonstradas publicamente durante uma sessão apoiada pela iniciativa europeia Sistema de Incentivos à Investigação e Desenvolvimento (SIID) – Investigação e Desenvolvimento (I&D) Empresarial – Operações Individuais. A plataforma resultante demonstrou uma integração eficaz entre os componentes da framework eProcess, incluindo o Edoclink, Form Builder, Process Monitor e Process Designer, confirmando o papel do OnlineDesk como ponto de entrada modular e escalável para a automação de processos.

Palavras-chave: OnlineDesk, eProcess framework, REST API, SOAP, Kafka, JSON, BPM

Abstract

This thesis presents the development and integration of several architectural enhancements to OnlineDesk, a component of the eProcess framework, an enterprise-level platform developed by Link Consulting to support the design, execution, and monitoring of business processes. The framework integrates modular components for form handling, document lifecycle management, and process monitoring, offering a cohesive solution for Business Process Management (BPM). Within this framework, OnlineDesk facilitates structured data intake, form-based request submission, and interaction with other backend systems. The work was conducted within the context of a real-world company project focused on improving interoperability, maintainability, and extensibility by refactoring OnlineDesk's communication protocol with Edoclink from Simple Object Access Protocol (SOAP) to Representational State Transfer (REST) Application Programming Interfaces (APIs), integrating dynamic form support through Form Builder, introducing event-based lifecycle tracking via Kafka and Process Monitor, and implementing a JavaScript Object Notation based (JSON-based) import mechanism for form and service configuration portability.

These improvements were developed iteratively and validated through stakeholder feedback and continuous testing. They were later integrated into live pilot scenarios and publicly demonstrated during a session supported by the European SIID (from the portuguese *Sistema de Incentivos à Investigação e Desenvolvimento*) – I&D (from the portuguese *Investigação e Desenvolvimento*) *Empresarial – Operações Individuais* initiative. The resulting platform demonstrated effective integration across components of the eProcess framework, including Edoclink, Form Builder, Process Monitor, and Process Designer, and confirmed OnlineDesk's role as a modular and scalable entry point for process automation.

Keywords: OnlineDesk, eProcess framework, REST API, SOAP, Kafka, JSON, BPM

Table of Contents

Originalidade e Direitos de Autor ..... **iii**

Dedicatória ..... **iv**

Agradecimentos ..... **v**

Declaração sobre a Utilização de Inteligência Artificial ..... **vi**

Resumo ..... **vii**

Abstract ..... **viii**

List of Figures ..... **xiii**

List of Acronyms ..... **xvii**

1. Introduction **1**

2. Company Characterization **4**

3. Modern Approaches to BPM: The eProcess Framework and OnlineDesk **6**

3.1. eProcess Architecture: Essential Components **7**

3.1.1. Edoclink: Document Management and Process Execution 8

3.1.2. Form Builder: Customizable Form Design for Data Input 8

3.1.3. Process Monitor: Tracking and Analytics for Process Insights 9

3.2. Customer Interaction Portals and Document Management in BPM **9**

3.2.1. Existing Customer Interaction Portals: Functionalities and Limitations 9

3.2.2. Document Management Solutions in BPM 11

3.2.3. Unique Positioning of OnlineDesk in eProcess 11

3.2.4. Real-Time Status Tracking 12

3.2.5. Secure Document Handling and Compliance 12

3.2.6. Comparative Analysis: Addressing Gaps in Traditional Solutions 12

3.3. The Role of OnlineDesk in the eProcess Framework **13**

3.4. Leveraging Ontology, AI, and Process Mining in eProcess **14**

3.4.1. Ontology-Driven Process Specification 14

3.4.2. Artificial Intelligence and Process Mining 14

3.5. Summary **15**

4. Requirements and Workflow Integration **16**

4.1. Refactoring Edoclink Communication Provider **16**

4.2. Integrating Form Builder into OnlineDesk **17**

4.3. Integrating OnlineDesk with Process Monitor **18**

4.4. Enabling Configuration Portability Through JSON Imports **19**

4.5. Summary **19**

5. Methodology **21**

5.1. Introduction to Agile Methodologies **21**

5.2. Tools Used in the Project **22**

5.2.1. Trello for Task Management 23

5.2.2. Git for Version Control 24

5.3. Development workflow **25**

5.3.1. Collaboration and Feedback Loops 25

5.4. Summary **27**

6. Architecture **28**

6.1. Architectural Overview **28**

6.2. Core Components **30**

6.2.1. OnlineDesk 30

6.2.2. Edoclink 32

6.2.3. Form Builder 33

6.2.4. Process Monitor 34

6.2.5. Process Designer 35

6.2.6. eProcess Factory 36

6.3. OnlineDesk Architecture and Internal Concepts **37**

6.4. Technologies 42
6.4.1. SOAP vs. REST: Refactoring Communication Protocols 43
6.4.2. Angular: Building Dynamic Frontend Interfaces 44
6.4.3. .NET: Ensuring Backend Reliability and Scalability 46
6.4.4. Middleware and Security 48

6.5. Summary 50

7. Implementation 51

7.1. Refactoring SOAP to REST API 51
7.1.1. Purpose of the Refactoring 52
7.1.2. Mapping SOAP Methods to REST Endpoints 53
7.1.3. Refactoring Process 54
7.1.4. Updating System Dependencies 57
7.1.5. Challenges and Solutions 58
7.1.6. Summary 59

7.2. Integration with the Form Builder 59
7.2.1. Purpose of the Integration 60
7.2.2. Establishing the Configuration for Form Builder Connectivity 61
7.2.3. Service for Form Builder API Communication 62
7.2.4. API Endpoint in OnlineDesk 65
7.2.5. Frontend Modifications 66
7.2.6. Handling Form Builder Templates in OnlineDesk 67
7.2.7. Plug-in for Form Builder 69
7.2.8. Challenges and Solutions 76
7.2.9. Summary 77

7.3. Integration with the Process Monitor 77
7.3.1. Purpose of the Integration 78
7.3.2. Testing Kafka Message Delivery to Process Monitor 79
7.3.3. Kafka Integration via EMonitorProvider 80
7.3.4. Triggering the Kafka Provider via OnlineDesk's Monitoring Service 85
7.3.5. Frontend Integration for the EMonitorProvider 87
7.3.6. Event-Based Communication with Process Monitor 92

7.3.7. Challenges and Solutions 94
7.3.8. Summary 95

7.4. Importing Services and Forms via JSON 96
7.4.1. Purpose of the Implementation 96
7.4.2. Frontend Implementation of Service Import 97
7.4.3. Backend Implementation of Service Import 102
7.4.4. Frontend Implementation of Form Import 104
7.4.5. Backend Implementation of Form Import 109
7.4.6. Challenges and Solutions 112
7.4.7. Summary 113

7.5. Summary 113

8. Demonstration and Real-World Validation 114

8.1. Demonstration Use Cases 114
8.1.1. Efetuar Reclamação (Submit a Complaint) 115

8.2. Feature Validation 116

8.3. Evidence of Usage 117

9. Conclusion 120

Bibliography 122

Annexes 128

List of Figures

Figure 1 - eProcess Vision [2] 7
Figure 2 - CRM Market Share [10] 10
Figure 3 - OnlineDesk frontpage 13
Figure 4 - Edoclink communication provider 17
Figure 5 - Form Builder Integration 18
Figure 6 - Process Monitor Integration 18
Figure 7 - Service and Form Configuration Sharing 19
Figure 8 - Trello Board 24
Figure 9 - Azure DevOps work item linked with Git commits 25
Figure 10 - Main Project Milestones 26
Figure 11 - eProcess High-Level Architecture Diagram [2] 29
Figure 12 - OnlineDesk Interface 31
Figure 13 - Edoclink Interface 32
Figure 14 - Form Builder Interface 34
Figure 15 - Process Designer Interface 36
Figure 16 - OnlineDesk Frontend and Backend Architecture 38
Figure 17 - OnlineDesk Request 39
Figure 18 - OnlineDesk Form 40
Figure 19 - OnlineDesk Services 41
Figure 20 - OnlineDesk Key Concepts Relationships 42
Figure 21 - SOAP Dataflow [39] 43
Figure 22 - Rest API Model [42] 44
Figure 23 - Angular Architecture [46] 45
Figure 24 - .NET Framework Architecture [49] 47
Figure 25 - OnlineDesk Login Interface 49
Figure 26 - Transition from SOAP-based to REST API between OnlineDesk and Edoclink 53
Figure 27 - SOAP to REST Mapping 54
Figure 28 - GetEdocBooksList() SOAP-Based Method 55

Figure 29 - GetEdocBooksList() REST-Based Method 56
Figure 30 - GetBooksList() SOAP-Based Method Invocation 57
Figure 31 - GetBooksList() REST-Based Method Invocation 58
Figure 32 - Integration Flow Between OnlineDesk and Form Builder 60
Figure 33 - Form Builder Configurations 62
Figure 34 - GetAccessTokenAsync() Method 63
Figure 35 - GetFormsAsync() Method 64
Figure 36 - LinkFormBuilder API Endpoint 65
Figure 37 - Form Creation 66
Figure 38 - Add API Endpoint 67
Figure 39 - PutFormularioDadosGerais API Endpoint 68
Figure 40 - GetFormularioDadosGerais API Endpoint 69
Figure 41 - Form Builder Plug-in Addition Interface 70
Figure 42 - Execute() Method 71
Figure 43 - OnlinedeskConfigService 72
Figure 44 - removeInvalidChars() Method 73
Figure 45 - generatePDF() Method 74
Figure 46 - convertBlobToBae64() Method 74
Figure 47 - submitForm() Method 75
Figure 48 - Integration Flow Between OnlineDesk and Process Monitor 78
Figure 49 - Kafka ProducerConfig 79
Figure 50 - Kafka Message 80
Figure 51 - IEMonitorProvider Interface 81
Figure 52 - Provider General Configurations 82
Figure 53 - Provider Form Configurations 83
Figure 54 - Kafka Message 83
Figure 55 - Kafka Producer 84
Figure 56 - Message Serialization and Delivery 84
Figure 57 - IEMonitorService Interface 85
Figure 58 - SendMessage() Method 86

Figure 59 - Backoffice Main Dashboard 87
Figure 60 - BackofficeHomeComponent Tiles 88
Figure 61 - Proccess Monitor Route 89
Figure 62 - Provider Definition General Data 89
Figure 63 - Provider Definition Configurations 90
Figure 64 - getMonitorProvider() Method 90
Figure 65 - ProvidersResolver 91
Figure 66 - Form Providers 91
Figure 67 - Form Providers Configuration 92
Figure 68 - EventConstants 93
Figure 69 - SendMessageEMonitor Event 94
Figure 70 - EMonitorDataConsumer Consume() Method 94
Figure 71 - JSON Import Workflow Across eProcess Components 97
Figure 72 - "Importar Serviço" Button 97
Figure 73 - Service Json File Structure 98
Figure 74 - selectFicheiro() Method 99
Figure 75 - handleServicoImport() Method 100
Figure 76 - handleServicoImport() Method 101
Figure 77 - importar() Method 101
Figure 78 - Importar Endpoint 102
Figure 79 - Service Validation 103
Figure 80 - Add or Update Service 103
Figure 81 - "Importar Formulário" Button 104
Figure 82 - Form Json File Structure 105
Figure 83 - selectFicheiro() Method 106
Figure 84 - Form Type Identification 106
Figure 85 - TipoFormularioMap 106
Figure 86 - Required Fields Validation 107
Figure 87 - Match Form Type 107
Figure 88 - Match Service 108

Figure 89 - Match Name 108
Figure 90 - importar() Method 108
Figure 91 - Importar Endpoint 109
Figure 92 - Form Validation 109
Figure 93 - Add Form and Service 110
Figure 94 - Add Provider 111
Figure 95 - Integration Flow of the “Efetuar Reclamação” Use Case 116

List of Acronyms

AI	Artificial Intelligence
AOT	Ahead-of-Time
API	Application Programming Interfaces
BPM	Business Process Management
CRM	Customer Relationship Management
CEJ	Centro de Estudos Judiciários
DEMO	Design & Engineering Methodology for Organizations
DGAJ	Direção-Geral da Administração da Justiça
DGPJ	Direção-Geral da Política de Justiça
DGRSP	Direção-Geral de Reinserção e Serviços Prisionais
ESTG	Escola Superior de Tecnologia e Gestão
GDPR	General Data Protection Regulation
gRPC	gRPC Remote Procedure Call
HTTP	Hypertext Transfer Protocol
I&D	Investigação e Desenvolvimento
IGFEJ	Instituto de Gestão Financeira e Equipamentos da Justiça
IGSJ	Inspeção-Geral dos Serviços de Justiça
INESC	Instituto de Engenharia de Sistemas e Computadores
INMLCF	Instituto Nacional de Medicina Legal e Ciências Forenses
IPL	Instituto Politécnico de Leiria
IRN	Instituto dos Registos e Notariado
JSON	JavaScript Object Notation
NLP	Natural Language Processing
PDF	Portable Document Format
REST	Representational State Transfer
SGMJ	Secretaria-Geral do Ministério da Justiça
SIID	Sistema de Incentivos à Investigação e Desenvolvimento
SLA	Service Level Agreements
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SSO	Single Sign-On

WSDL	Web Services Description Language
XML	eXtensible Markup Language

1. Introduction

This thesis documents the development of several key features in OnlineDesk, a component of the eProcess framework, a modular BPM platform developed by Link Consulting to support the design, execution, and monitoring of digital workflows across organizations. The framework integrates reusable components for structured data intake, form handling, document lifecycle management, process orchestration, and real-time monitoring, enabling scalable and interoperable process automation.

Within this framework, OnlineDesk acts as a bridge between external users and internal process engines, providing a central entry point for form-based submissions, document exchange, and lifecycle tracking across different stages of request processing.

Throughout the course of this project, multiple architectural improvements were implemented to enhance OnlineDesk's flexibility, maintainability, and integration with other components of the eProcess framework.

These improvements included the refactoring of OnlineDesk's legacy SOAP-based integration with Edoclink into a modern REST API, the integration with Form Builder to support dynamic form management, the creation of a Kafka-based event emission system for lifecycle monitoring with Process Monitor, and the implementation of a JSON-based import mechanism for replicating form and service configurations across environments.

The work was carried out as part of a company-hosted project aligned with the goals of the SIID – I&D *Empresarial – Operações Individuais* program and the Digital Europe initiative. Throughout the development process, iterative validation and stakeholder feedback guided architectural and implementation choices, ensuring that the proposed features addressed real operational needs. The solutions were tested both in isolation and as part of real-world workflows, including a public demonstration held in May 2025.

This chapter concludes by presenting the structure of the document and how each chapter contributes to the characterization of the overall development and evaluation of the OnlineDesk integration project within the eProcess framework.

It begins with chapter 2 (Company Characterization), which provides an overview of Link Consulting, the organizational context of the project, and the positioning of the eProcess framework within the company's solutions.

Chapter 3 (Modern Approaches to BPM: The eProcess Framework and OnlineDesk), introduces the technological background and architectural principles behind the eProcess framework. It provides an overview of the core components and explains how these elements work together to support process automation. The chapter also explores eProcess's relevance to modern BPM, highlighting how it integrates ontology-driven modeling, artificial intelligence (AI), and process mining techniques. Special attention is given to the role of OnlineDesk as a user-facing component responsible for data intake, request initiation, and coordination with other backend systems.

Chapter 4 (Requirements and Workflow Integration) describes the specific technical requirements and integration workflows addressed throughout the project. It frames the core functionalities that guided implementation work, including improvements to communication protocols, configuration flexibility, and system interoperability.

Chapter 5 (Methodology) presents the methodological approach adopted during development. It outlines the agile practices used, tools selected for project management and version control and justifies the iterative workflow and development that shaped the project lifecycle.

Chapter 6 (Architecture) focuses on the architecture of the eProcess framework, explaining and detailing the roles of each component and the technologies used to ensure system performance, extensibility, and interoperability.

Chapter 7 (Implementation) presents the technical implementation of the features developed throughout the project. Each subsection details one major enhancement, including the migration from SOAP to REST, the integration with Form Builder, the Kafka-based communication with Process Monitor, and the JSON import functionality.

Chapter 8 (Demonstration and Real-World Validation) demonstrates the platform's practical validation, presenting two real-world pilot use cases developed and executed as part of a public demonstration event. It also includes stakeholder testimonies, and confirmation of usage within a list of entities preparing to adopt the platform.

Finally, Chapter 9 (Conclusion) concludes the document with reflections on the outcomes of the project and potential directions for future development and enhancement of the eProcess framework.

2. Company Characterization

Link Consulting is a Portuguese information technology and consulting company headquartered in Lisbon. It was founded as a spin-off from the Instituto de Engenharia de Sistemas e Computadores (INESC) and Instituto Superior Técnico, and has since specialized in the development of modular technological solutions, including proprietary software and integration services. The company's focus areas include digital transformation, public administration modernization, and enterprise system interoperability.

Operating internationally, Link Consulting maintains project engagements and commercial activity across Europe, the Middle East, Africa, and Latin America. Countries with established presence or partnerships include Brazil, Belgium, Ghana, Saudi Arabia, and the United Kingdom. The company's legal form is that of a private limited company (*sociedade por quotas*), and its core activity lies in the information technology and consulting sector.

Its activities are distributed across several business units within the Link Group, including Link Consulting (focused on research and integration), Link 101, LinkCom, and FS.Data. The company combines internally developed software componentes with consulting services in enterprise architecture, governance, risk and compliance, software quality assurance, DevOps, project management, and digital marketing. Its clients span geographically and vary in size, from public institutions to large private enterprises, and include public sector institutions, healthcare providers, financial entities, telecommunications and media companies, mobility providers, utilities, and infrastructure operators. Projects often address digitalization, service optimization, and integration of legacy systems with modern platforms.

One of the company's most recent initiatives is the eProcess project, developed under the scope of the SIID – I&D *Empresarial – Operações Individuais* program and co-funded by the European Union. The eProcess framework is a BPM platform designed to support hyperautomation across organizations by enabling end-to-end design, execution, and monitoring of processes across distributed environments and heterogeneous workflow engines. It integrates multiple components developed by Link Consulting, including OnlineDesk, Edoclink, Form Builder, Process Designer, Process Monitor, and Process Factory, each contributing to a specific phase of the process lifecycle.

This thesis was developed within the scope of the eProcess project and focuses specifically on the enhancement of OnlineDesk. The work is aligned with the goals of the broader initiative, particularly regarding maintainability, scalability, and reuse of process assets across entities. The developed features were validated through integration into two pilot scenarios, in a public demonstration in May 2025, which showcased the platform's applicability to real-world contexts.

3. Modern Approaches to BPM: The eProcess Framework and OnlineDesk

The rapid evolution of BPM systems reflects an ongoing demand for solutions that can streamline complex workflows, improve customer interactions, and manage documents effectively. Traditional BPM approaches, while effective at structuring and automating internal processes, often lack the flexibility, integration, and advanced functionality required to meet modern organizational needs. In response, next-generation BPM frameworks, like eProcess, have emerged to bridge these gaps by integrating artificial intelligence, ontology-driven process specifications, and process mining techniques.

Within the eProcess framework, OnlineDesk serves as a crucial customer interaction portal, enabling seamless communication, document submission, and request tracking for end users. To provide a comprehensive understanding of OnlineDesk's role, this section explores the latest developments in BPM, customer interaction portals, document management, and relevant technological innovations. This review highlights the advancements that make OnlineDesk a distinctive component of eProcess and sets the foundation for a deeper examination of its contributions to efficient business process automation.

Section 3.1 (eProcess Architecture: Essential Components) details the core architecture of eProcess, highlighting essential components like Edoclink, Form Builder, and Process Monitor, which work together with OnlineDesk to enable efficient process automation. Next, section 3.2 (Customer Interaction Portals and Document Management in BPM), examines the current landscape of BPM customer portals and document management, identifying the limitations of traditional systems, and introducing how OnlineDesk within eProcess uniquely addresses these challenges. The following section, 3.3 (The Role of OnlineDesk in the eProcess Framework), focuses on OnlineDesk itself, explaining its integration with other components to deliver a powerful experience for managing customer interactions and documents. Afterward, section 3.4 (Leveraging Ontology, AI, and Process Mining in eProcess) explores the main technologies of eProcess, such as ontology, artificial intelligence, and process mining, and highlights their impact on OnlineDesk’s capabilities and adaptability. Finally, section 3.5 (Summary) concludes the chapter by summarizing the broader eProcess vision as an integrated BPM solution, highlighting the collaboration

between OnlineDesk and the other components that together define a modern approach to BPM.

3.1.eProcess Architecture: Essential Components

The eProcess framework is a proprietary platform developed by Link Consulting. It serves as an integrated ecosystem for managing business processes, customer interactions, and document workflows. Designed to address limitations in some traditional BPM systems, eProcess combines specialized components into a unified architecture that ensures seamless integration and automation across various organizational functions, like deliver seamless document management, process execution, and customer interaction. Its design enables organizations to automate workflows, enhance decision-making through data insights, and improve customer engagement via efficient service portals [1].

As shown in Figure 1, the eProcess framework integrates key components, each fulfilling specific roles within process definition, development, and execution.

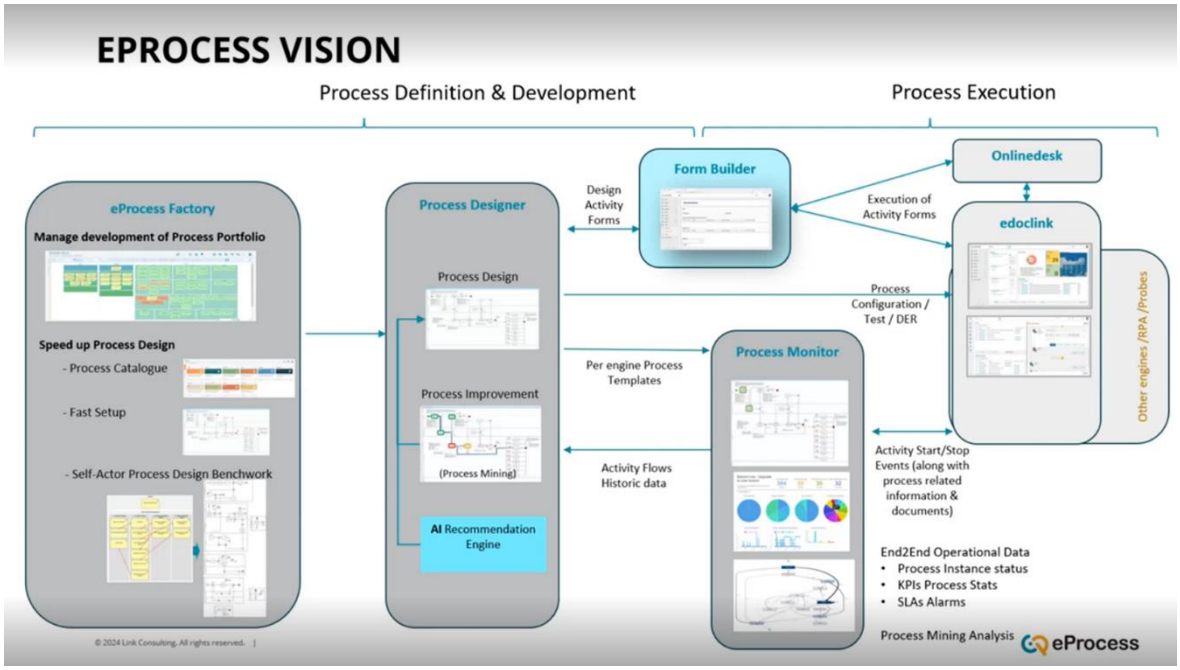


Figure 1 - eProcess Vision [2]

The eProcess Factory and Process Designer handle process creation and optimization, leveraging features like process mining analytics and AI-driven recommendations. The Form Builder enables the creation of customized activity forms tailored to specific business needs which are executed through OnlineDesk to manage customer interactions. Edoclink handles the document lifecycle, ensuring secure storage, classification, and compliance.

Finally, Process Monitor tracks process metrics, provides analytics, and ensures adherence to Service Level Agreements (SLA). The following sections provide a high-level overview of the components most relevant to OnlineDesk.

3.1.1. Edoclink: Document Management and Process Execution

Edoclink serves as the primary engine within eProcess for managing documents and executing processes. As a document management system, Edoclink supports the entire lifecycle of documents, from creation and classification to secure storage and retrieval. This functionality is essential for OnlineDesk, as it enables efficient handling of customer-submitted documents, ensuring that all forms, requests, and associated documents are securely stored and easily accessible for processing.

Moreover, Edoclink integrates directly with OnlineDesk, allowing for seamless document submission and tracking. For example, when a customer submits a form through OnlineDesk, Edoclink manages the document's classification, versioning, and storage, ensuring compliance with data security standards. This integration ensures that customer requests are managed efficiently and tracked accurately throughout the process lifecycle, creating a streamlined and secure document handling experience within OnlineDesk.

3.1.2. Form Builder: Customizable Form Design for Data Input

The Form Builder within eProcess allows organizations to create and customize forms that are essential for capturing data in process activities. As the primary tool for designing and executing forms, Form Builder supports OnlineDesk by enabling tailored data collection interfaces for different service requests and activities. This flexibility allows administrators to configure forms based on specific customer needs, ensuring that data is captured accurately and efficiently.

In the OnlineDesk interface, these customized forms provide the default structure for customer interactions, whether for submitting new requests, providing feedback, or tracking request status. By leveraging Form Builder, OnlineDesk can support a variety of service request types, making it a versatile and adaptable customer interaction portal that caters to diverse business needs.

3.1.3. Process Monitor: Tracking and Analytics for Process Insights

Process Monitor is the component within eProcess responsible for capturing the start and end events of each activity, along with providing comprehensive analytics on process performance. This monitoring capability is critical for understanding how processes unfold over time and for identifying areas for improvement. In the context of OnlineDesk, Process Monitor plays a supporting role by tracking key metrics related to customer-submitted requests, such as processing times, completion rates, and adherence to SLA.

The data gathered by Process Monitor can be used to generate insights that improve OnlineDesk's efficiency and responsiveness. For example, if processing times for certain types of requests are consistently high, adjustments can be made to streamline these workflows. Additionally, by tracking SLA and other performance metrics, Process Monitor helps ensure that customer service standards are met, enhancing overall user satisfaction and enabling data-driven improvements to OnlineDesk operations [3].

3.2.Customer Interaction Portals and Document Management in BPM

With the increasing emphasis on customer-centric service models, customer interaction portals have become essential in BPM systems. These portals serve as the primary interface for client engagement, enabling customers to initiate and monitor service requests, communicate with support teams, and manage document submissions. However, despite their widespread adoption, many traditional BPM portals still struggle to meet the full spectrum of modern demands, particularly around customization, real-time updates, and integrated document handling. This section examines the current context of customer interaction portals and document management solutions within BPM, emphasizing how OnlineDesk within the eProcess framework addresses these challenges.

3.2.1. Existing Customer Interaction Portals: Functionalities and Limitations

Customer portals in BPM systems, such as Salesforce Service Cloud [4], Microsoft Dynamics 365 Customer Service [5], SAP Customer Experience [6], Oracle CX Service [7], and Adobe Experience Manager [8], provide users with essential tools for managing service requests and tracking case statuses. These portals allow customers to submit queries, interact with customer support representatives, and receive updates on their case progress. For instance, Salesforce Service Cloud is widely used for its customizable workflows and integration with other Salesforce services, which allows organizations to build customer-

specific workflows [4]. Similarly, Microsoft Dynamics 365 offers a robust solution for case management, enabling multi-channel customer service interactions and integrations with Microsoft's broader ecosystem [5]. In addition, SAP Customer Experience provides strong customer service capabilities, focusing on omnichannel interactions and integration with SAP's enterprise resource planning (ERP) systems [6]. Oracle CX Service emphasizes personalized customer experience through AI-driven recommendations and analytics [7]. Meanwhile, Adobe Experience Manager stands out for its focus on content management and digital marketing, allowing businesses to integrate customer support seamlessly into their branding efforts [8].

The choice of these platforms as examples, as shown in Figure 2, is due to their significant market share and reputation as industry leaders in customer interaction portals. According to recent market analyses, Salesforce holds approximately 20% to 23% of the Customer Relationship Management (CRM) market share, making it the largest CRM vendor globally. Microsoft Dynamics 365 is another top player, ranking consistently among the top five CRM solutions and known for its strong integration capabilities and multi-channel service support [9] [10].

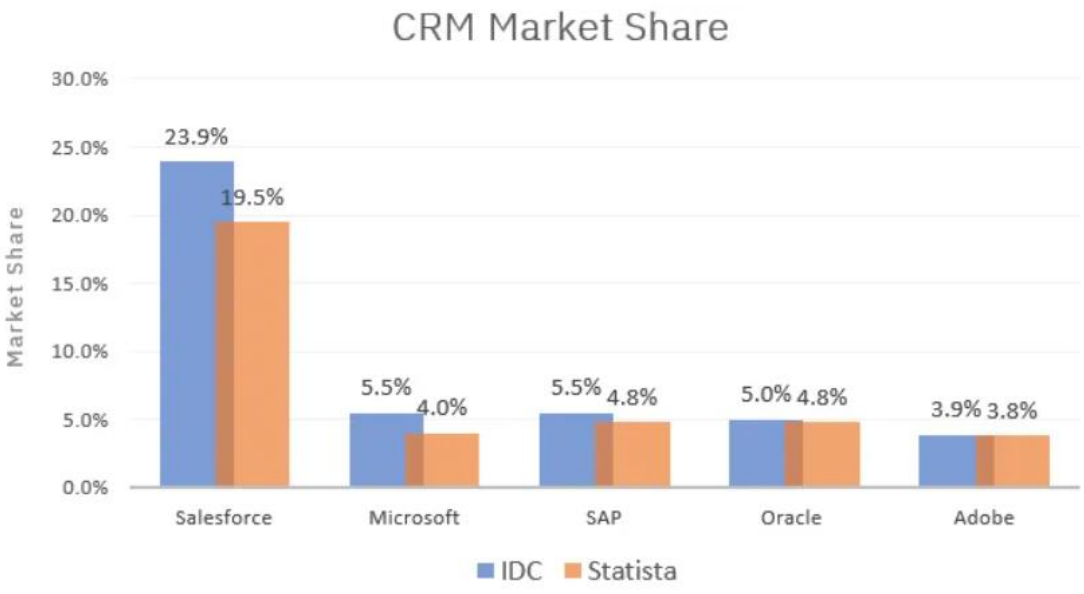


Figure 2 - CRM Market Share [10]

These platforms represent widely adopted solutions, providing a relevant benchmark for discussing the functionalities and limitations of existing customer interaction portals.

Despite their strengths, however, these platforms typically lack the depth of customization needed to handle highly variable service requests across different industries. Moreover, while they provide options for attaching documents to service requests, they often lack advanced document lifecycle management features such as automatic classification, versioning, and compliance-based access control [1] [4] [5] [6] [7] [8].

3.2.2. Document Management Solutions in BPM

Document management capabilities are central to BPM, particularly in systems where customer requests require secure handling, processing, and tracking of submitted documents. Solutions like IBM Filenet and OpenText Content Suite have become foundational in many organizations for managing documents within business processes. IBM's Filenet, for example, supports large-scale content management, with capabilities for secure storage, indexing, and compliance management [11]. OpenText's Content Suite similarly provides robust document management tools, with advanced search, auditing, and records management features, making it a strong choice for organizations with high compliance requirements [12].

While some BPM systems offer document management features integrated with customer interaction portals, this level of integration is not always seamless or comprehensive. In many cases, standalone document management systems, like Filenet and OpenText, are used alongside BPM systems, creating a disconnect that can introduce delays. For example, documents may need to be manually transferred between the customer portal and the document management system for processing. Such separation can also lead to security and access challenges, as documents are handled across multiple platforms, increasing the likelihood of information silos and inconsistencies [3] [11] [12].

3.2.3. Unique Positioning of OnlineDesk in eProcess

The OnlineDesk component within eProcess stands out in the BPM landscape by offering a fully integrated approach to customer interaction and document management. Designed specifically to bridge the gaps found in traditional BPM systems, OnlineDesk incorporates real-time status tracking, secure document handling, and a user-friendly interface, directly addressing modern requirements for customer service [13]. This integration is achieved through its close relationship with other eProcess components, particularly Edoclink for document lifecycle management and Process Monitor for tracking request statuses.

3.2.4. Real-Time Status Tracking

One of the primary limitations in traditional BPM portals is the lack of real-time updates. OnlineDesk, through its connection with Process Monitor, allows customers to track the progress of their service requests and view status updates at each stage. This transparency is a significant improvement over legacy systems, which may provide only periodic updates or require manual status checks by support staff. By offering real-time insights, OnlineDesk enhances customer satisfaction, as users remain informed and engaged throughout the process.

3.2.5. Secure Document Handling and Compliance

Data privacy and security are crucial for any BPM system that handles sensitive customer documents. With its integration with Edoclink, OnlineDesk ensures that submitted documents are stored and managed securely, incorporating features such as version control, access restrictions, and automated classification. These capabilities address compliance requirements like General Data Protection Regulation (GDPR), which mandates strict handling and storage of personal data. Unlike traditional portals, where document handling may be limited or partially outsourced to separate systems, OnlineDesk ensures that all documents remain within the secure confines of the eProcess framework, reducing risk and ensuring compliance [1] [3].

3.2.6. Comparative Analysis: Addressing Gaps in Traditional Solutions

Traditional BPM portals and standalone document management systems play critical roles in managing business processes, but they can often face challenges in integration, customization, and real-time processing, which can create information silos, leading to inefficiencies and delays in service delivery [3] [14]. Similarly, BPM portals that lack deep integration with document management systems require manual interventions, increasing the risk of errors and slowing down response times [15].

eProcess addresses these limitations through the integration of its components, each contributing to an efficient and cohesive workflow. Through this integration, eProcess tries to eliminate the inefficiencies of traditional systems by automating workflows, improving data consistency, and enabling faster, more transparent service delivery [1].

3.3. The Role of OnlineDesk in the eProcess Framework

Within eProcess, OnlineDesk serves as the dedicated customer interaction portal. It plays a crucial role in bridging the gap between the organization and its clients by managing the submission, tracking, and communication of service requests and documents. Through OnlineDesk, users can securely submit forms, track the status of their requests, and receive updates, making it a key component for enhancing customer satisfaction and service transparency [1].

OnlineDesk relies on the surrounding eProcess components to achieve its full potential. For example, it interacts with Edoclink to manage the lifecycle of submitted documents, ensuring they are stored, classified, and processed securely, with Form Builder to customize and create forms used in service requests, which provides a flexible and user-friendly interface for data input/output or Process Monitor to track key metrics related to customer requests, enabling the system to provide real-time status updates and manage SLA. Figure 3, exemplifies how OnlineDesk integrates these functionalities into a cohesive and user-friendly platform.

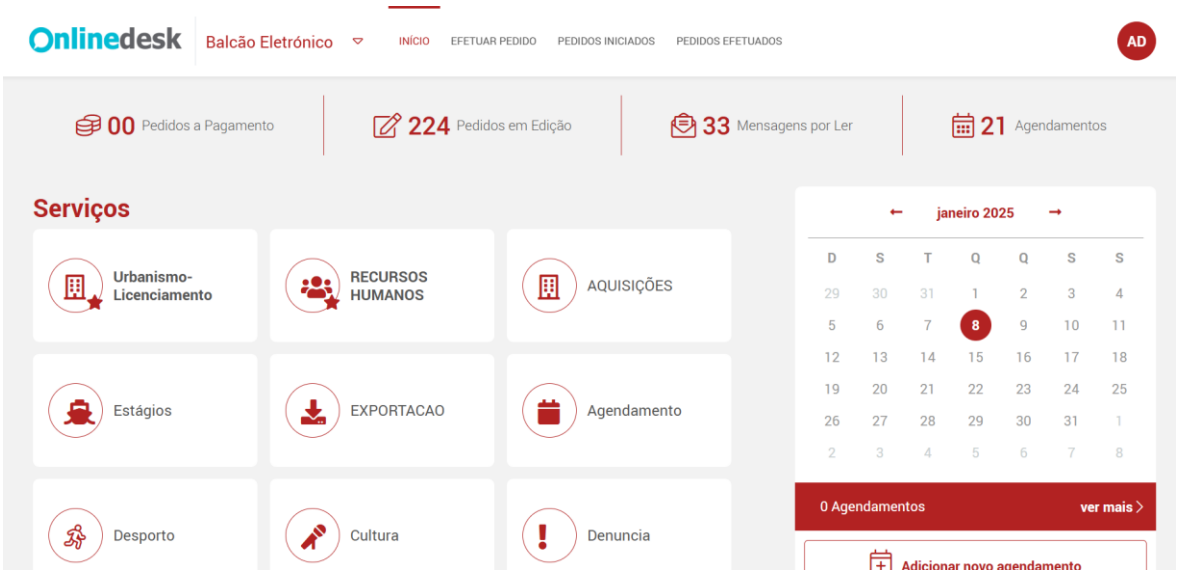


Figure 3 - OnlineDesk frontpage

While OnlineDesk is customer-facing, it draws strength from the integrated eProcess framework to deliver a seamless experience. The synergy between OnlineDesk and the other eProcess components is critical, as it enables end-to-end automation and document management that would otherwise require extensive manual intervention.

3.4.Leveraging Ontology, AI, and Process Mining in eProcess

The eProcess framework leverages advanced technologies like ontology-driven specifications, artificial intelligence (AI), and process mining to overcome limitations in traditional BPM systems. These concepts ensure that each component of eProcess, including OnlineDesk, operates cohesively and efficiently, even in complex, multi-user environments. This section explores these advanced concepts, focusing on how they enhance OnlineDesk's functionality by improving process consistency, document management accuracy, and customer responsiveness.

3.4.1. Ontology-Driven Process Specification

Ontology-driven specifications provide a structured approach to representing business processes by defining entities, relationships, and attributes that make up each process. This approach helps ensure that processes are modeled consistently across eProcess, supporting standardization in how workflows are structured and managed. Within the OnlineDesk component, ontology-driven specification is critical for reliable document handling and classification. By creating a shared vocabulary and framework, ontology helps minimize ambiguities in process definitions, making it easier to design processes that meet diverse customer needs while maintaining alignment with organizational standards [1].

Ontology in BPM is further supported by academic work such as Dietz and Mulder's "Enterprise Ontology" and frameworks like Design & Engineering Methodology for Organizations (DEMO), which establish methodologies for using ontology to streamline organizational processes and improve communication across [1] [16]. These principles applied in eProcess enhance OnlineDesk's ability to manage and interpret customer-submitted information in a structured, reliable manner, reducing errors and inconsistencies.

3.4.2. Artificial Intelligence and Process Mining

Artificial intelligence and process mining play a transformative role in modern BPM by automating routine tasks, improving decision-making, and providing insights into process performance. In eProcess, AI supports document classification, allowing OnlineDesk to automatically categorize and route customer submissions based on content analysis. This reduces manual sorting errors and ensures that documents are processed efficiently. Process mining complements this functionality by analyzing historical data to identify patterns and optimize workflows, helping OnlineDesk adapt to evolving customer needs [1] [3].

For example, AI algorithms such as Natural Language Processing (NLP) can automatically analyze the content of submitted forms to detect keywords or intents, enabling OnlineDesk to streamline routing and processing based on document type. Process mining tools also enable eProcess to continuously improve its workflows by identifying bottlenecks and suggesting optimizations. These improvements directly benefit OnlineDesk, allowing it to provide faster response times and more personalized service by dynamically adjusting processes in response to data insights [1].

Together, ontology, AI, and process mining contribute to an adaptable and responsive eProcess framework. For OnlineDesk, these technologies ensure that customer interactions are handled efficiently and consistently, supporting the overall objective of providing a high-quality experience for end users.

3.5.Summary

This chapter highlighted how the eProcess framework represents a modern approach to BPM by addressing limitations in traditional systems. Through its modular and integrated design, eProcess combines technologies like ontology, AI, and process mining with its components to deliver automation, decision-making, and customer engagement.

OnlineDesk plays an important role as the customer interaction portal, integrating closely with components like Edoclink for document management, Form Builder for customizable data collection, and Process Monitor for real-time tracking. These integrations streamline workflows, reduce inefficiencies, and enhance both internal operations and customer-facing services.

By uniting these components into a cohesive system, eProcess provides an adaptable BPM solution capable of meeting the evolving demands of modern organizations. Its vision emphasizes efficiency, flexibility, and transparency, ensuring that customer interactions and organizational processes are managed within a unified structure.

The work developed throughout this project contributed to advancing this vision by reinforcing the integration between OnlineDesk and other components of the framework, enabling the platform to meet its intended goals.

4. Requirements and Workflow Integration

Improving the integration and functionality of OnlineDesk within the eProcess framework required addressing a set of evolving technical needs, focused on communication, configuration management, and system interoperability. This section introduces the core project requirements that guided the implementation work detailed in later chapters.

The chapter begins with section 4.1 (Refactoring Edoclink Communication Provider), which focuses on replacing legacy SOAP based services [17] with REST APIs [18] to improve performance and scalability. Section 4.2 (Integrating Form Builder into OnlineDesk) introduces a dynamic approach to form handling by integrating Form Builder, allowing forms to be defined and rendered externally. In section 4.3 (Integrating OnlineDesk with Process Monitor), the focus shifts to real-time observability, where lifecycle events in OnlineDesk are tracked and communicated to Process Monitor. Lastly, section 4.4 (Enabling Configuration Portability Through JSON Imports) presents a mechanism for importing structured services and form configurations, improving interoperability across eProcess environments.

It is important to note that the company initially provided these requirements at a high level and refined iteratively throughout the project, allowing the requirements to be further refined and granularized as the project progresses and specific challenges arise. This iterative methodology aligns with the evolving nature of the project, providing room to adapt the solution to unforeseen complexities or opportunities for improvement.

The following sections describe each requirement in greater detail and explain their importance in the broader development of a cohesive and scalable eProcess framework.

4.1. Refactoring Edoclink Communication Provider

One of the foundational requirements is modernizing the communication protocol between Edoclink and OnlineDesk. Currently, this interaction relies on SOAP-based services, which, while functional, introduce inefficiencies and integration challenges. SOAP is a protocol that provides standardized messaging for web services but is often criticized for its complexity and verbosity, which can hinder performance in modern applications [17]. Refactoring this communication to use a REST API aims to address these limitations,

providing a more modern, scalable, and secure protocol [18]. In practice, this change transforms how requests are exchanged between the two components. Figure 4 illustrates the intended interaction model between OnlineDesk and Edoclink once the RESTful communication layer is in place. This approach not only streamlines communication but also lays the groundwork for future scalability and interoperability within the eProcess framework.

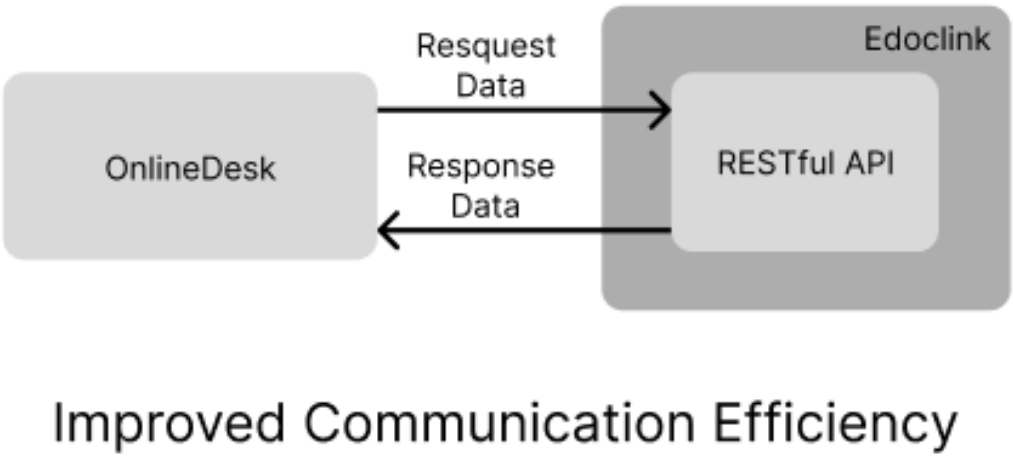


Figure 4 - Edoclink communication provider

4.2.Integrating Form Builder into OnlineDesk

To improve the functionality of data collection operations, OnlineDesk is expected to be extended to support integration with Form Builder, a dynamic and configurable form creation tool that is part of the eProcess framework. This integration is going to allow OnlineDesk to leverage externally managed templates from Form Builder while still supporting other native form types, such as predefined HTML or web-based forms.

When a Form Builder form is selected, OnlineDesk should dynamically load the template and redirect users to complete it in the appropriate environment. Upon submission, both structured data and a Portable Document Format (PDF) representation are intended to be returned to OnlineDesk for storage and further processing, maintaining consistency with existing workflows. As depicted in Figure 5, this integration is expected to enable seamless interaction with external forms and ensure accurate data capture, enabling users to interact with the system effectively. By bringing this capability into OnlineDesk, the system should become more adaptable, meeting diverse needs while maintaining a user-friendly experience.

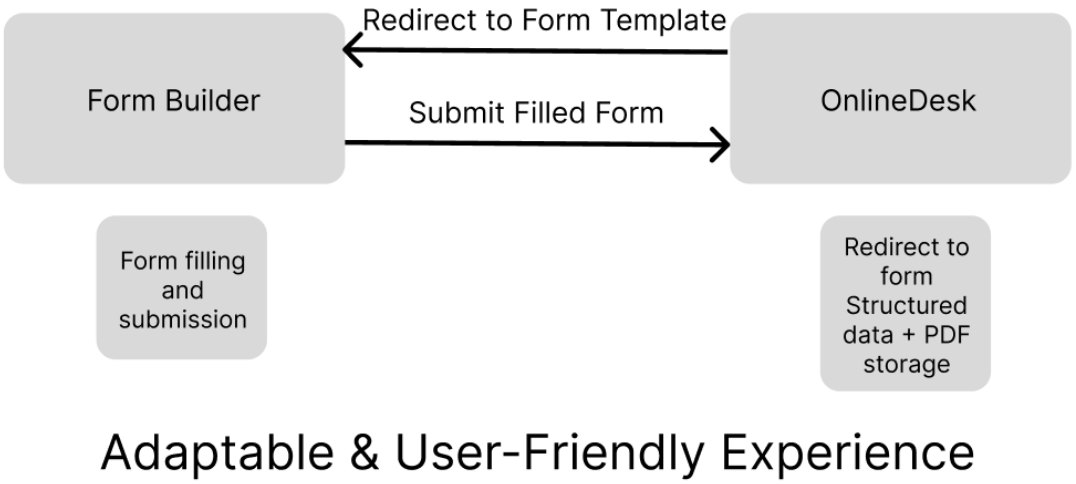


Figure 5 - Form Builder Integration

4.3.Integrating OnlineDesk with Process Monitor

To support real-time observability of request execution within the eProcess framework, one of the project's requirements involves integrating OnlineDesk with the Process Monitor component. The Process Monitor is responsible for tracking the execution lifecycle of requests across various states. As illustrated in Figure 6, to support this functionality, OnlineDesk is expected to emit updates to Process Monitor whenever relevant request lifecycle events occur. This integration is intended to improve transparency and supports analytics that provide operational insights across the eProcess framework.

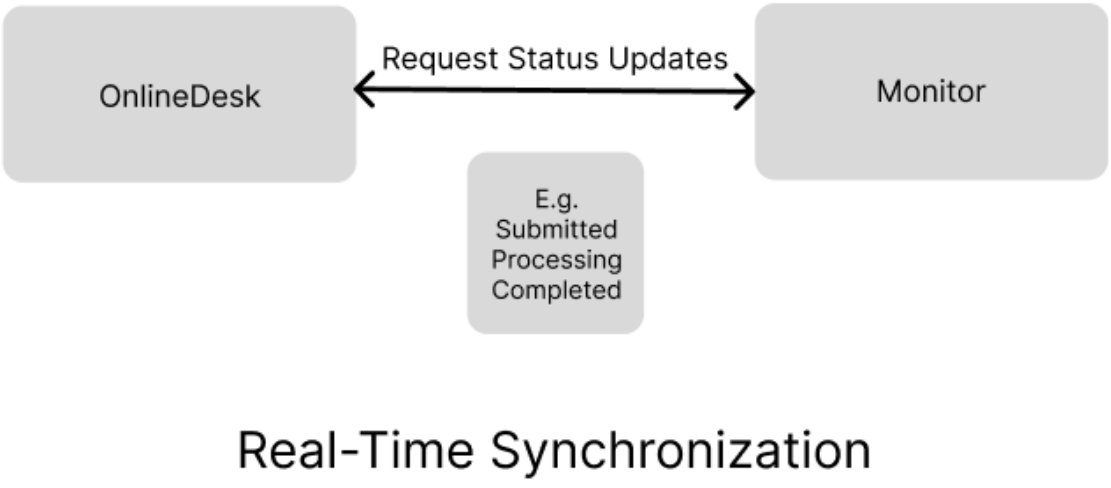


Figure 6 - Process Monitor Integration

4.4.Enabling Configuration Portability Through JSON Imports

To promote configuration portability and interoperability across the eProcess framework, another requirement of this project is to enable OnlineDesk to support the import of services and forms directly from structured JSON files. This functionality is intended to allow entity configurations, such as hierarchical services or customized form definitions, to be exported from one system and imported into another, eliminating the need for repetitive manual setup.

The import feature is expected to rely on a standardized JSON schema that defines the fields, structure, and relationships between services and forms.

As depicted in Figure 7, this capability aims to facilitate consistent and synchronized configurations across different OnlineDesk instances. The JSON-based structure should ensure that the process remains both human-readable and machine-processable, enabling integrations with other eProcess components or export tools.

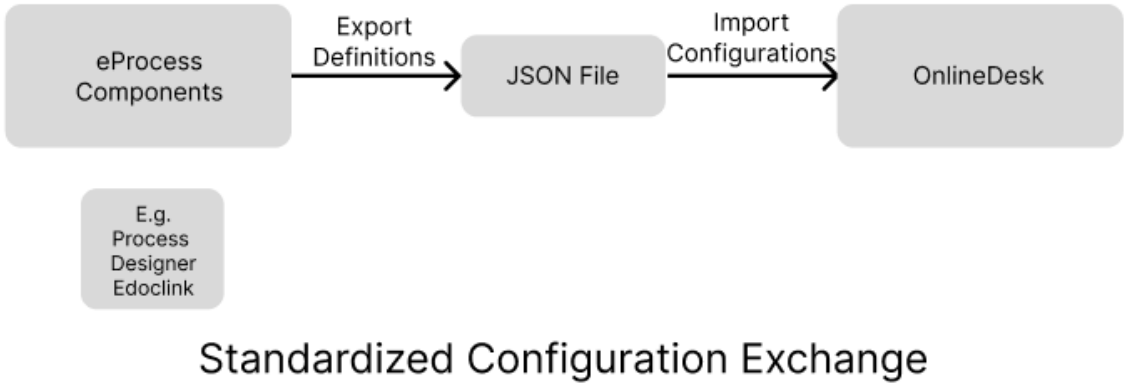


Figure 7 - Service and Form Configuration Sharing

4.5.Summary

The requirements defined in this chapter outline the foundational improvements necessary to align OnlineDesk with the broader goals of the eProcess framework. The modernization of communication with Edoclink through REST APIs is expected to improve scalability and maintainability. The integration of Form Builder aims to enable dynamic, configurable form handling, while the connection to Process Monitor is intended to support real-time tracking of workflow execution. Finally, the planned introduction of a JSON-based import mechanism is designed to simplify the sharing and replication of services and forms across environments, thereby promoting interoperability and reducing manual setup.

Together, these requirements establish a clear path for a more modular, transparent, and adaptable BPM platform.

5. Methodology

Developing a robust and scalable project requires not only technical expertise but also a well-defined and adaptable workflow. This chapter details the methodologies, tools, and strategies employed to manage the project effectively.

The chapter begins with section 5.1 (Introduction to Agile Methodologies), providing an overview of agile principles and the selection of Kanban as the primary framework for its adaptability and iterative approach. This is followed by section 5.2 (Tools Used in the Project), showcasing how Trello and Git were integrated into the workflow to manage tasks and maintain version control effectively. The development process is detailed in section 5.3 (Development workflow), which explains how tasks were broken down, tracked, and iterated upon while incorporating regular feedback from stakeholders, including company members and the project advisor. Finally, section 5.4 (Summary) summarizes how the combination of agile principles, Kanban, Trello, and Git ensured a structured and efficient project development process.

Through these sections, this chapter illustrates how the chosen methodology ensured collaboration, supported iterative development, and aligned the project with its goals.

5.1. Introduction to Agile Methodologies

Agile methodologies are an iterative approach to project management and software development that prioritize flexibility, collaboration, and continuous improvement. Unlike traditional waterfall models, which rely on rigid, sequential phases, agile frameworks encourage breaking down projects into smaller, manageable tasks that can be completed in iterative cycles [19]. Each iteration produces a functional deliverable, allowing teams to adapt quickly to changing requirements or feedback. Agile's focus on regular communication and iterative progress ensures a more dynamic and responsive workflow, particularly valuable in software development projects where needs and goals may evolve over time [20].

For this project, agile methodologies were especially suitable due to the iterative nature of the requirements and the evolving scope of tasks. With the need to refine communication protocols, integrate new components, and ensure seamless workflows between OnlineDesk and other eProcess components, an adaptable approach was essential. Additionally, the

development team consisted of a single developer (for the requirements described), making the clarity, prioritization, and iterative progress central to managing the workload effectively. Collaboration with other actors, including company members who provided guidance and support when doubts or challenges arose, was vital to overcoming obstacles and ensuring the project stayed aligned with organizational goals. Agile's flexibility allowed the project to progress incrementally, incorporating feedback at every stage and ensuring that development aligned with technical and organizational requirements [21].

Within the agile spectrum, Kanban was selected as the primary framework for managing the project. Kanban offers several advantages that align with the needs of this development process. By visualizing the workflow on a Kanban board, tasks could be categorized into stages such as To Do, In Progress, and Done [22]. This visualization provided clear visibility into the project's progress at any given time. Furthermore, Kanban's principle of limiting work in progress helped to maintain focus on priority tasks and avoid the inefficiencies of multitasking [23]. This approach not only improved task prioritization but also ensured steady and manageable progress throughout the project lifecycle.

Kanban also proved particularly advantageous in managing the modular nature of the platform. Since the project involved enhancing distinct components, being able to track and manage tasks independently within the same board helped maintain clarity while allowing for cohesive development. The flexibility of Kanban also enabled the project to respond effectively to evolving requirements.

Agile methodologies, and specifically Kanban, provided the necessary structure and adaptability for this project. By fostering continuous improvement, enabling clear visualization of tasks, and maintaining focus through work-in-progress limits, the methodology ensured a smooth and efficient development process tailored to the project's unique requirements [24].

5.2. Tools Used in the Project

The successful execution of this project required tools that complemented the agile methodology and streamlined tasks and version control. Trello and Git were chosen for their simplicity, efficiency, and ability to support iterative development. Trello provided a visual platform for managing tasks, while Git ensured robust version control across multiple

repositories. Together, these tools helped maintain clarity, organization, and collaboration throughout the project.

Although the development team consisted of only one person, collaboration with other actors played a crucial role in the project's success. These actors included company members who provided technical guidance, clarified requirements, and assisted in resolving challenges, as well as the project advisor, who offered critical feedback and direction. The use of Trello and Git supported this collaboration by making progress visible and facilitating the sharing and review of work.

5.2.1. Trello for Task Management

Trello is a web-based project management tool that implements the Kanban methodology. It organizes tasks as cards on a board, with each card representing a specific task. Boards are divided into columns, typically labeled to reflect stages of progress, such as To Do, In Progress, and Done. The ability to drag and drop cards between columns provides an intuitive way to visualize task progression and prioritize work effectively [25].

In this project, Trello was used to manage tasks across different components and requirements, serving as the primary tool for tracking progress and aligning work with project milestones. A dedicated board was created to reflect the workflow, and tasks were categorized into columns that represented their current status.

Each card included a description, checklists, and due dates, ensuring transparency and accountability. This setup allowed for clear prioritization and helped to keep the workflow aligned with project milestones [26].

By integrating Trello into the workflow, the project leveraged an intuitive and flexible tool that aligned with the iterative development cycles inherent in agile methodologies [19]. Trello's ability to visualize tasks and monitor progress in real time made it indispensable for maintaining clarity and focus throughout the development process.

Figure 8 illustrates the Trello board used in the project, showcasing how tasks were organized, tracked, and moved across workflow stages during implementation.

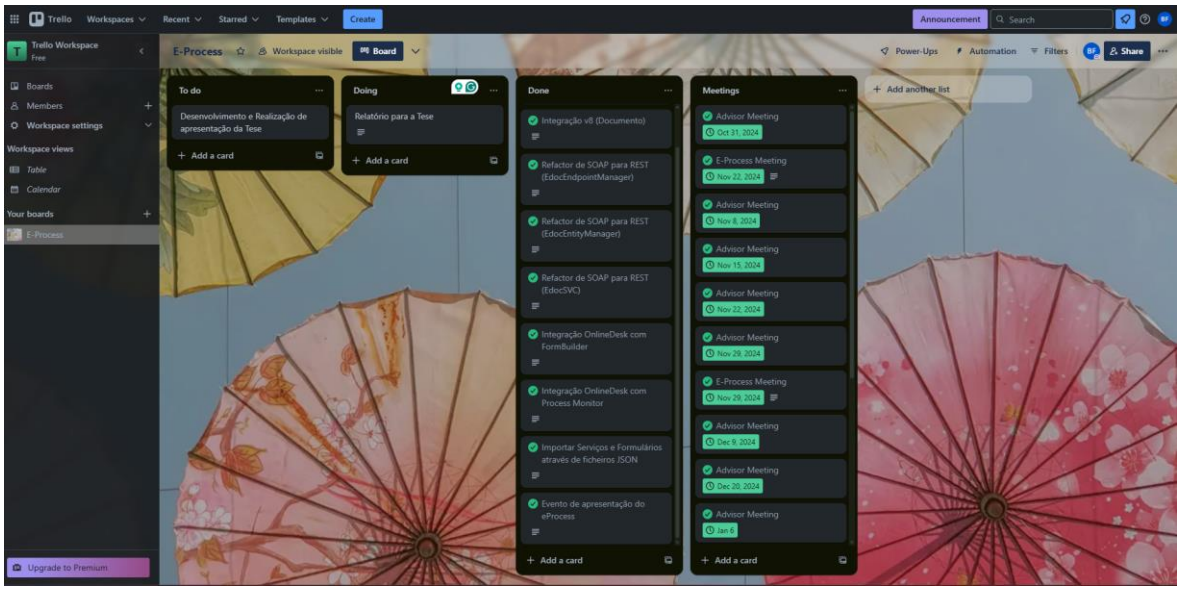


Figure 8 - Trello Board

5.2.2. Git for Version Control

Git is a distributed version control system widely used for tracking changes in code. It allows developers to collaborate efficiently by managing code in branches, enabling concurrent development without conflicts. In this project, Git was used to maintain a structured and collaborative workflow, with each branch corresponding to a specific requirement or feature.

It is important to understand how Git supports version control. Changes to the codebase are tracked as commits, each of which represents a snapshot of progress. Commits are stored in branches, which are isolated workspaces that allow developers to experiment or implement features without affecting the main codebase. Once work on a branch is complete, it can be merged into the main branch after testing and review [27].

Git's integration with platforms like Azure DevOps further enhanced collaboration, providing a centralized location for repositories and facilitating features like pull requests and code reviews. To improve visibility and maintain traceability between work items and code changes, Azure DevOps was used to associate commits with specific tasks. Each requirement was linked to a corresponding work item, allowing progress to be tracked and reviewed in context. Figure 9 shows an example of this integration, highlighting a work item with its related commit history and associated repository activity. This workflow ensured that the codebase remained organized and stable throughout the iterative development process [28].

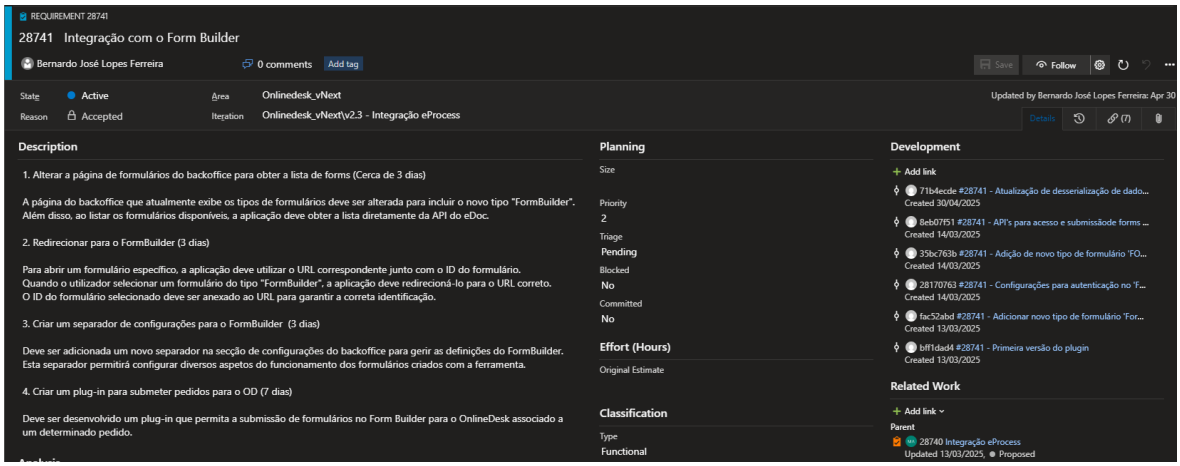


Figure 9 - Azure DevOps work item linked with Git commits

5.3. Development workflow

The development workflow for this project was designed to align with agile principles, focusing on iterative progress, collaboration, and seamless integration of tools. The process was structured to ensure that tasks were broken down into manageable pieces, and stakeholders were actively involved in providing feedback and guidance.

5.3.1. Collaboration and Feedback Loops

Collaboration with stakeholders was a crucial aspect of the development workflow. Agile practices highlight the importance of regular communication and feedback loops, which enhance project alignment and improve outcomes [26]. Stakeholders included individuals within the company, such as team members and project supervisors, as well as the project advisor. Regular feedback sessions were held to review progress, discuss challenges, and refine requirements. These sessions were typically conducted and served as a platform for presenting completed work, identifying areas for improvement, and aligning the next steps. These feedback loops were instrumental in maintaining the project's alignment with stakeholder expectations and enhancing the quality of the deliverables.

These iterative feedback loops and agile practices led to several tangible benefits. Task clarity improved, as complex requirements were broken down into smaller, manageable items, enabling a clear understanding of what needed to be done and allowing for more precise planning and execution, making it easier to track progress and address issues proactively. Regular delivery of functional increments allowed stakeholders to review and refine progress continuously. Additionally, the flexibility of Kanban and agile methodologies proved valuable in responding to unforeseen challenges, maintaining a steady

progress and responding effectively to changing circumstances while ensuring alignment with project goals.

To better visualize the progression of the project, Figure 10 presents a timeline of key milestones. Each milestone corresponds to a major feature implementation or review checkpoint, where stakeholder input played a crucial role in guiding the next phase of development.

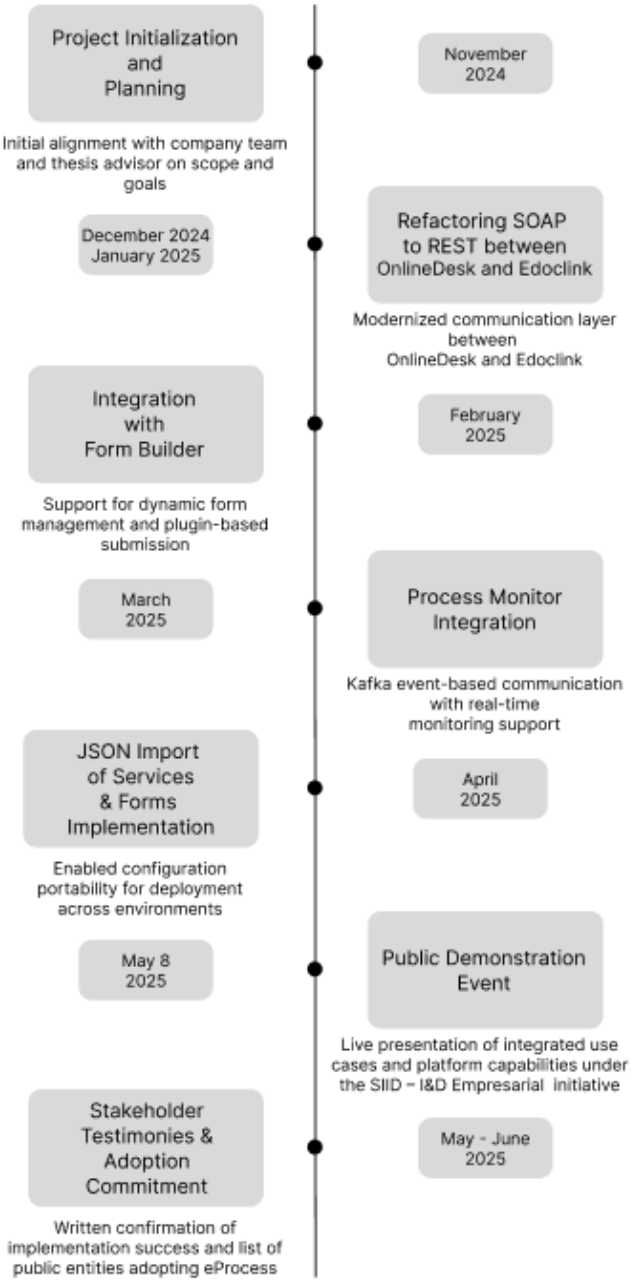


Figure 10 - Main Project Milestones

5.4.Summary

The successful implementation of this project relied on the effective application of agile principles, Kanban methodology, and tools like Trello and Git. Together, these elements provided a flexible and efficient framework, enabling steady progress despite evolving requirements and project complexities.

Agile principles fostered adaptability and iterative development, ensuring each increment aligned with project goals and incorporated stakeholder feedback. Kanban complemented this by offering a visual, structured approach to task management through Trello boards, which tracked progress in real-time, prioritized tasks, and maintained productivity by limiting work in progress.

Git supported the project with its robust version control, enabling efficient collaboration and maintaining a stable codebase. Its branching strategy streamlined development, while features like commit tracking and reversions enhanced resilience to challenges.

By integrating these methodologies and tools, the project achieved its objectives efficiently, supporting iterative development, clear prioritization, and collaboration across all stages.

6. Architecture

The architecture of any system serves as the foundation upon which its functionality, performance, scalability, maintainability, efficiency, adaptability, extensibility, security, and reliability are built. In the context of the eProcess framework, the architecture ensures the integration of its various components, enabling end-to-end process automation, the efficient management of business processes, customer interactions, and document workflows.

This chapter begins in section 6.1 (Architectural Overview), which introduces the high-level architecture of eProcess, highlighting the two main operational layers, definition and execution, and how they contribute to the platform's adaptability. Section 6.2 (Core Components) provides a detailed explanation of each major module, including their roles, interconnections, and contribution to the overall workflow. While all components are presented to offer a complete architectural overview, special attention is given to OnlineDesk, Edoclink, Form Builder, and Process Monitor, the modules directly involved in the implementation work developed in this project. Also, given OnlineDesk's central role across multiple integration scenarios, section 6.3 (OnlineDesk Architecture and Internal Concepts) offers a deeper examination of its internal architecture, data model, and processing flow. It clarifies key entities such as requests, forms, services, and providers, providing essential context for understanding the implementation decisions described in later chapters. Section 6.4 (Technologies) describes the technology stack behind OnlineDesk, outlining the rationale for selecting frameworks such as Angular and .NET, and explaining how REST APIs, authentication protocols, and middleware enable secure and efficient communication across the ecosystem. Finally, section 6.5 (Summary) concludes the chapter by revisiting the architectural principles and highlighting how the selected design supports integration, extensibility, and operational efficiency across the eProcess framework.

Through this architectural analysis, the chapter demonstrates how eProcess aligns with modern business requirements, providing a robust, scalable, and secure foundation for process automation and customer service excellence.

6.1. Architectural Overview

The eProcess framework is a structured platform designed to manage business processes efficiently, integrating various components that work together to define, develop, and

execute workflows. It integrates multiple components to facilitate process design, execution, monitoring, and document management, ensuring end-to-end operational efficiency. The architecture is designed to provide flexibility, scalability, and interoperability, supporting a wide range of business operations.

At its core, eProcess employs a distributed architecture where each component is responsible for a specific function within the system. The system follows a hybrid approach, combining service-oriented architecture (SOA) principles with modular microservices, facilitating interoperability and integration across various tools and subsystems.

As illustrated in Figure 11, the architecture is structured into two primary phases, each playing a distinct role in the system's functionality. The diagram illustrates the interaction between the system's core components, and highlights the structured flow from process definition to execution, showcasing how data and actions propagate through the different modules

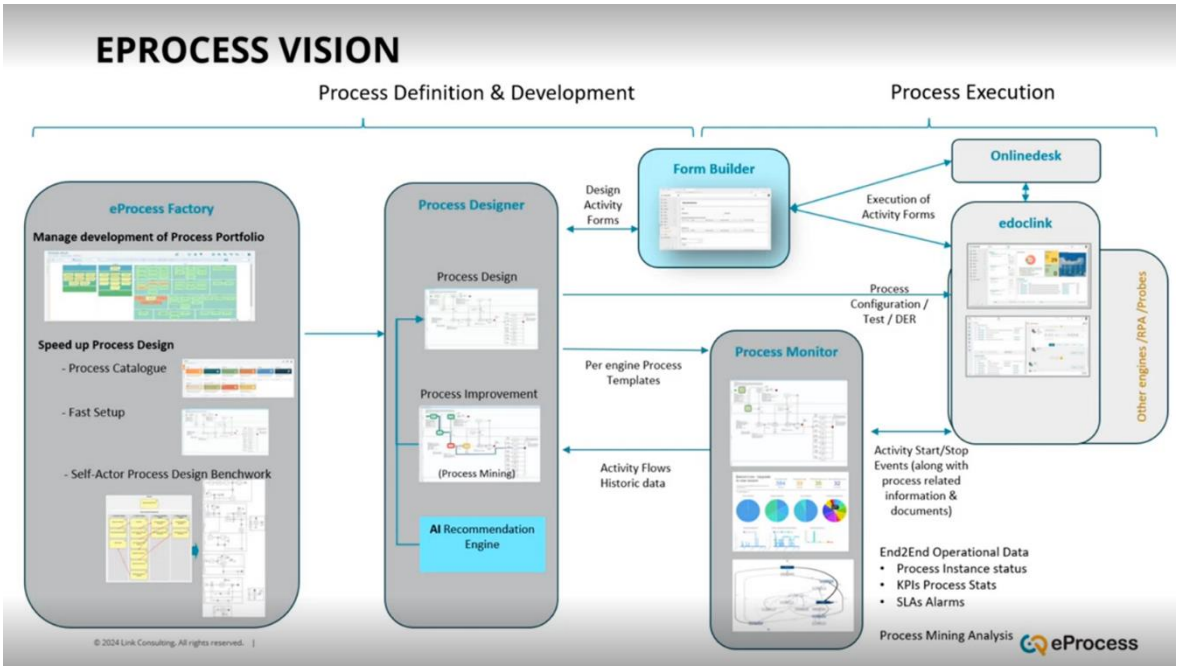


Figure 11 - eProcess High-Level Architecture Diagram [2]

The Process Definition and Development Layer focuses on supporting process modeling, automation, and optimization. This phase is powered by tools such as eProcess Factory, for managing the process portfolio, eProcess Designer for process modeling and improvement, and Form Builder for designing activity forms, which facilitate the creation and refinement of business processes, ensuring they are well-structured and efficient.

Complementing this, the Process Execution Layer is responsible for managing request execution, overseeing the document lifecycle, and enforcing compliance. This layer operates through key components such as OnlineDesk, which serves as an interface for users interacting with services, and Edoclink, which processes requests and is responsible for document lifecycle management, ensuring process execution. Additionally, it enables real-time tracking of process instances, monitors SLA compliance, and provides performance insights through the Process Monitor, ensuring continuous improvement and operational efficiency.

This structured approach ensures smooth coordination between process modeling, workflow execution, and performance monitoring, forming a cohesive and adaptable BPM solution [2].

6.2.Core Components

The eProcess architecture is built around modular components, each responsible for a specific function in process execution and management. These components work together to support process definition, document handling, request execution, and performance monitoring.

This section provides a detailed breakdown of the key components, focusing on OnlineDesk, Edoclink, Form Builder, and Process Monitor, along with supporting components like Process Designer and eProcess Factory.

6.2.1. OnlineDesk

OnlineDesk serves as an interface for users within the eProcess framework, functioning as a dedicated portal for managing customer interactions, submitting service requests, and tracking their progress. It streamlines communication between users and the backend services of eProcess by integrating with other core components such as Edoclink for document management and Process Monitor for tracking execution statuses.

Figure 12 provides an example of the OnlineDesk interface, showcasing how users can interact with the platform.

Figure 12 - OnlineDesk Interface

As a front-facing component, OnlineDesk enables end users to access various business processes while maintaining security and compliance. It provides an intuitive interface that facilitates service execution by allowing users to upload documents, fill in forms, and receive status updates in real-time. This interaction is essential for ensuring that business processes are not only executed efficiently but also provide transparency to stakeholders.

To achieve this, OnlineDesk is designed to support multiple authentication mechanisms, ensuring secure access control. It integrates with Keycloak for OAuth 2.0-based authentication [29] [30], supports Windows Authentication for enterprise environments [31], and provides local authentication options for standalone deployments [32]. This flexibility allows organizations to implement the security protocols that best fit their requirements while ensuring a seamless login experience.

Furthermore, OnlineDesk interacts with the Form Builder to allow dynamic form generation for different service requests. This integration enables users to complete and submit customized forms directly through the platform, ensuring that business-specific data is captured accurately. Additionally, OnlineDesk facilitates the automated processing of these requests by communicating with backend components such as Edoclink, which handles document lifecycle management, and Process Monitor, which ensures real-time status tracking [33].

6.2.2. Edoclink

Edoclink serves as the core document management and process execution engine within eProcess. It plays a pivotal role in handling workflows, ensuring that document-driven processes are managed efficiently from initiation to completion. As a workflow engine, Edoclink is responsible for executing predefined business processes, orchestrating tasks, and ensuring compliance with organizational policies.

One of Edoclink's primary functions is managing the full lifecycle of documents. It enables secure document storage, categorization, and retrieval, allowing users to track document modifications, maintain version control, and ensure compliance with data security policies. Additionally, it integrates structured and natural language search capabilities, enabling users to quickly locate relevant documents within business processes.

Figure 13 illustrates the Edoclink interface, showcasing its document management and workflow execution capabilities.

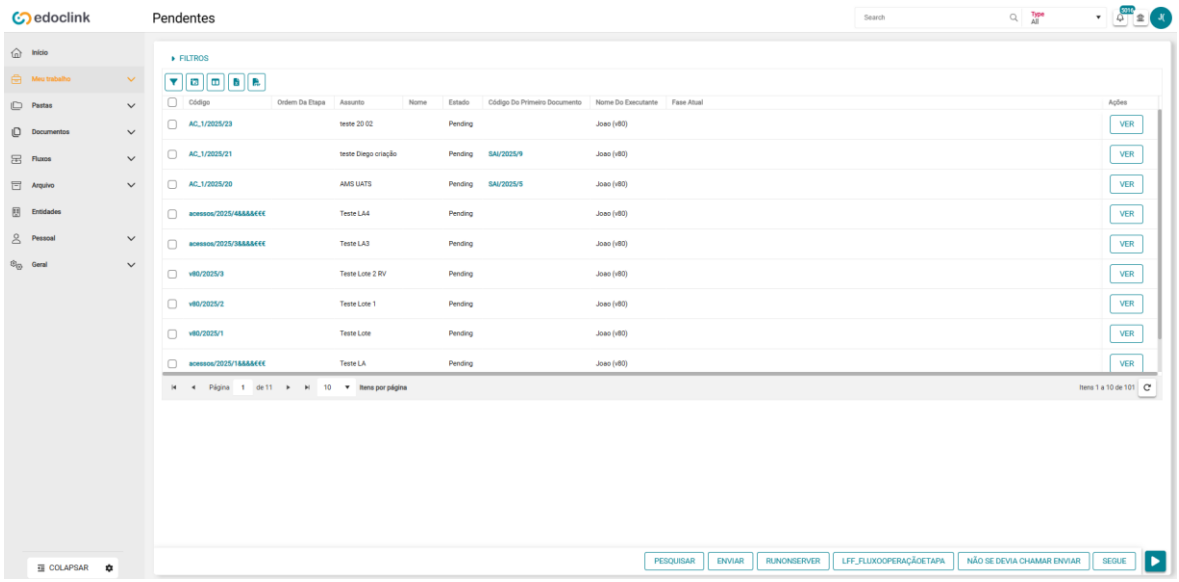


Figure 13 - Edoclink Interface

Edoclink operates in close collaboration with OnlineDesk to ensure seamless document submission and tracking. When a request is initiated through OnlineDesk, Edoclink plays a crucial role in processing all associated documents efficiently, maintaining a structured flow between user interactions and backend workflows. This integration allows for a streamlined experience, ensuring that documents are handled systematically and in accordance with the required processes.

Beyond its interaction with OnlineDesk, Edoclink also integrates with other key components of the eProcess framework. One such component is the Form Builder, which enables the dynamic creation of forms, allowing users to input relevant information during process execution. Another integral component is the Process Monitor, which provides real-time tracking of document-related process events. It logs execution history and offers insights into overall process performance, enhancing transparency and accountability within the system.

Through these integrations, Edoclink facilitates smooth execution of document workflows across the entire eProcess framework, improving operational efficiency while ensuring compliance with established protocols [34] [2].

6.2.3. Form Builder

Form Builder is a critical component of eProcess, enabling the creation and execution of dynamic forms used in business process activities. It serves as the primary tool for defining structured data input and output, ensuring that each step of a workflow can be customized based on the specific needs of a process.

One of its main functions is to facilitate the design of forms that can be integrated into business workflows. Through an intuitive interface, users can flexibly create forms that are linked to specific tasks, capturing and structuring varied data inputs, customizing field behaviors, and applying validations. This adaptability allows different use cases to define unique forms that are then embedded into the BPM workflow, ensuring that task execution is aligned with process requirements.

Figure 14 provides an example of the Form Builder interface, illustrating its role in designing structured forms for process automation.

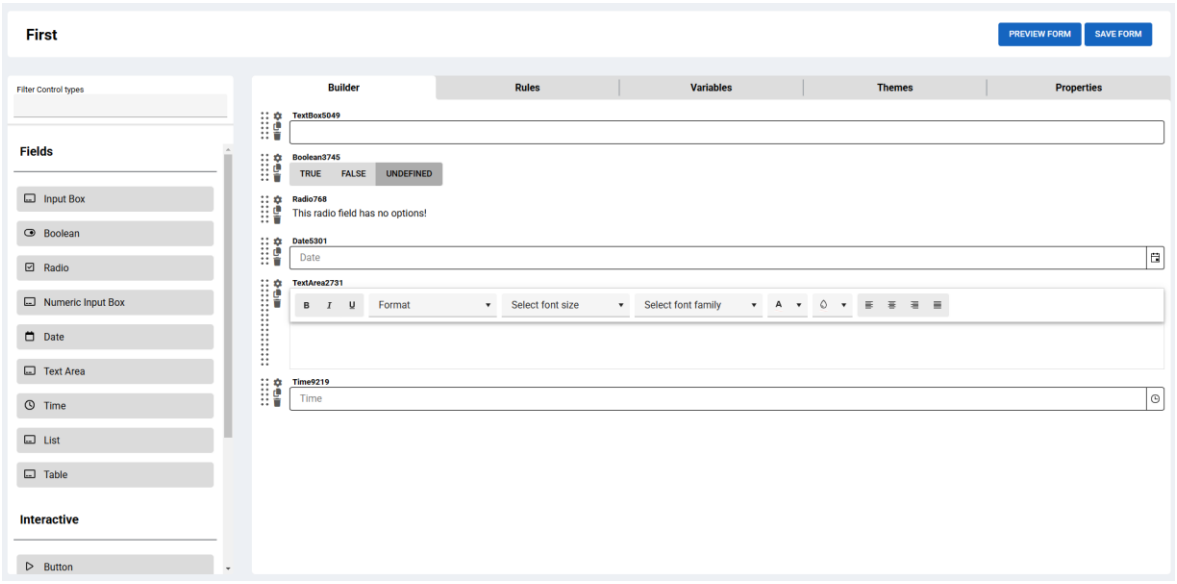


Figure 14 - Form Builder Interface

The integration between Form Builder and OnlineDesk is essential for maintaining a cohesive workflow experience. When users interact with a service request on OnlineDesk, they are presented with forms generated through Form Builder. This allows for an efficient data collection process that supports changes over time, for example, adjusting fields or layout based on evolving regulations or internal policies, while reducing manual intervention and minimizing errors.

Additionally, Form Builder is closely tied to Edoclink, ensuring that all document-related workflows are handled systematically. Forms created through Form Builder can be used to trigger automated actions in Edoclink, such as document approvals, signature requests, or validation processes. This integration reinforces the adaptability of the system, enhances automation and improves the efficiency of handling document-based workflows.

This integration within the eProcess framework ensures that Form Builder is a form creation tool to create structured interactions between users, documents, and process execution [35] [2].

6.2.4. Process Monitor

Process Monitor is a key component of the eProcess framework, responsible for collecting, analyzing, and visualizing real-time process execution data. It monitors the start and end of activities, tracks SLA compliance, and provides actionable insights into process performance.

At its core, Process Monitor records all significant process events, such as task initiation and completion, and stores this data in a structured format for further analysis. It is built on a modular architecture consisting of three primary components: a Message Reception Module, which receives notifications from workflow engines via Kafka, a Database, which stores event data and process models, and an Analysis Module, which processes this data using the PM4Py framework for process mining analytics. This structured approach allows organizations to maintain comprehensive audit trails and identify performance bottlenecks in real time. The system generates output in JSON format, integrating with Process Designer dashboards, ensuring that process managers always have up-to-date analytics for data-driven decision-making.

Process Monitor also issues alerts and notifications when SLA violations are at risk. By tracking predefined thresholds, the system proactively sends alerts to managers, enabling timely interventions and preventing workflow inefficiencies. Additionally, it supports comparative analysis across different process mining tools, allowing organizations to evaluate and optimize their process monitoring methodologies.

The integration of Process Monitor with OnlineDesk ensures that customer-facing interactions remain aligned with backend process execution. By continuously tracking task execution statuses and processing times, it allows OnlineDesk to provide accurate status updates to end-users, enhancing service transparency and reliability.

By providing detailed performance metrics and historical analysis, Process Monitor enables process managers to assess workflow efficiency over time. Its capabilities extend beyond real-time monitoring, allowing retrospective evaluations that support continuous process refinement [36] [2].

6.2.5. Process Designer

Process Designer serves as the central modeling tool within eProcess, enabling organizations to define, configure, and optimize their business processes. It provides a BPM-based interface that allows users to visually model workflows, define activity sequences, and establish decision logic. Figure 15 presents an example of a BPM workflow modeled within Process Designer, showcasing how business processes are structured and configured. Through its integration with workflow engines, Process Designer ensures that processes are correctly executed and monitored throughout their lifecycle.

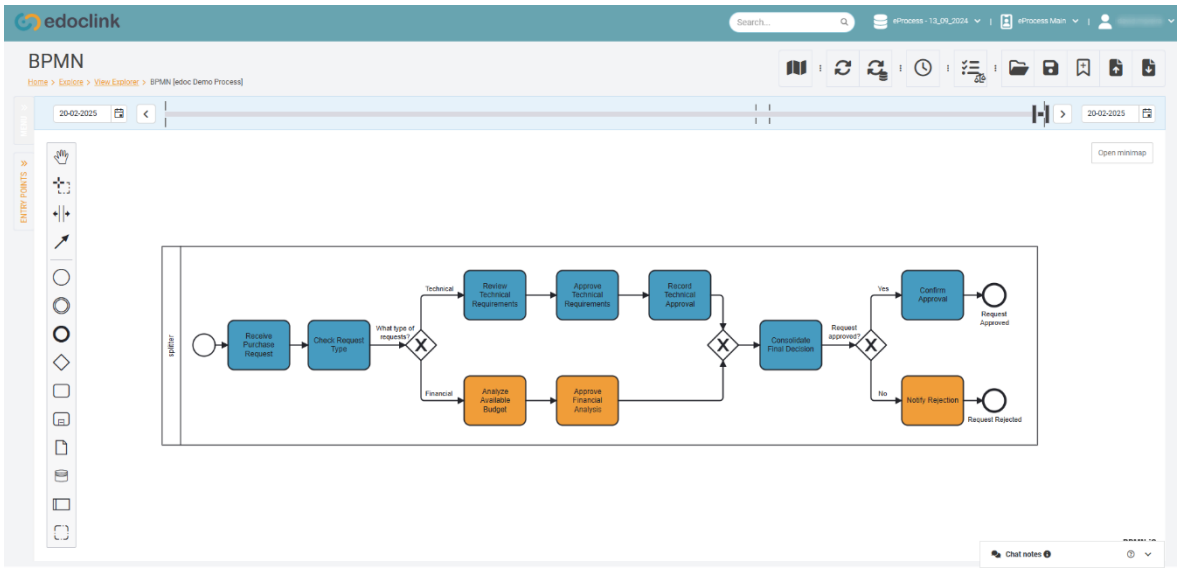


Figure 15 - Process Designer Interface

Another functionality of Process Designer is its integration with workflow execution engines. It supports task automation across different workflow environments, including Edoclink, which plays an important role in document lifecycle management.

To uphold performance and compliance standards, Process Designer incorporates SLA management, allowing organizations to track execution deadlines, monitor performance metrics, and ensure adherence to predefined service levels.

Furthermore, Process Designer works with Process Monitor, enabling real-time tracking of process execution. By sending structured event logs for analysis, it provides transparency into workflow performance and facilitates continuous process improvement [37] [2].

6.2.6. eProcess Factory

The eProcess Factory serves as a process portfolio management platform, designed to facilitate the deployment, optimization, and continuous improvement of workflows across various business areas. By providing a structured repository of reusable process templates, it enables organizations to rapidly implement workflows while ensuring consistency across operations.

One of its capabilities is the automation of process deployment through self-actor design tools, allowing for execution without extensive manual intervention.

Beyond deployment, the eProcess Factory plays a crucial role in continuous process refinement. By analyzing execution data, it can identify areas for improvement and

recommend workflow optimizations, ensuring that business processes evolve in alignment with operational needs and performance goals.

By reducing the time required to implement new workflows, maintaining uniformity across processes, and driving the adoption of optimized business models, the eProcess Factory enhances operational efficiency and promotes a more agile, data-driven approach to business process management [2].

6.3. OnlineDesk Architecture and Internal Concepts

While section 6.2.1 (OnlineDesk) provided a high-level overview of OnlineDesk's role within the eProcess framework, given the central role OnlineDesk plays in multiple integration scenarios presented throughout this document, this section presents a more detailed analysis of its internal architecture and operating principles. OnlineDesk operates as a modular platform composed of multiple frontends on the client side, supported by a .NET Core backend capable of delegating tasks to configurable providers responsible for external communications, payment handling, fee calculations, and more.

OnlineDesk's architecture is divided into two primary layers, the frontend, built with Angular, and the backend, implemented in .NET Core. The frontend is organized into a set of microfrontends on the client side, each responsible for distinct interaction contexts. The backend exposes a suite of RESTful APIs responsible for data persistence, authentication, and external integrations. A key architectural feature of the backend is its use of providers, which are modular, pluggable components that encapsulate communication logic with external systems such as document repositories, fee calculations, payment services, notification engines, or monitoring systems. These providers allow OnlineDesk to remain loosely coupled and highly extensible, as new functionalities can be introduced without modifying core logic.

This architectural split, and the interplay between the frontend microapps and the backend, is illustrated in Figure 16.

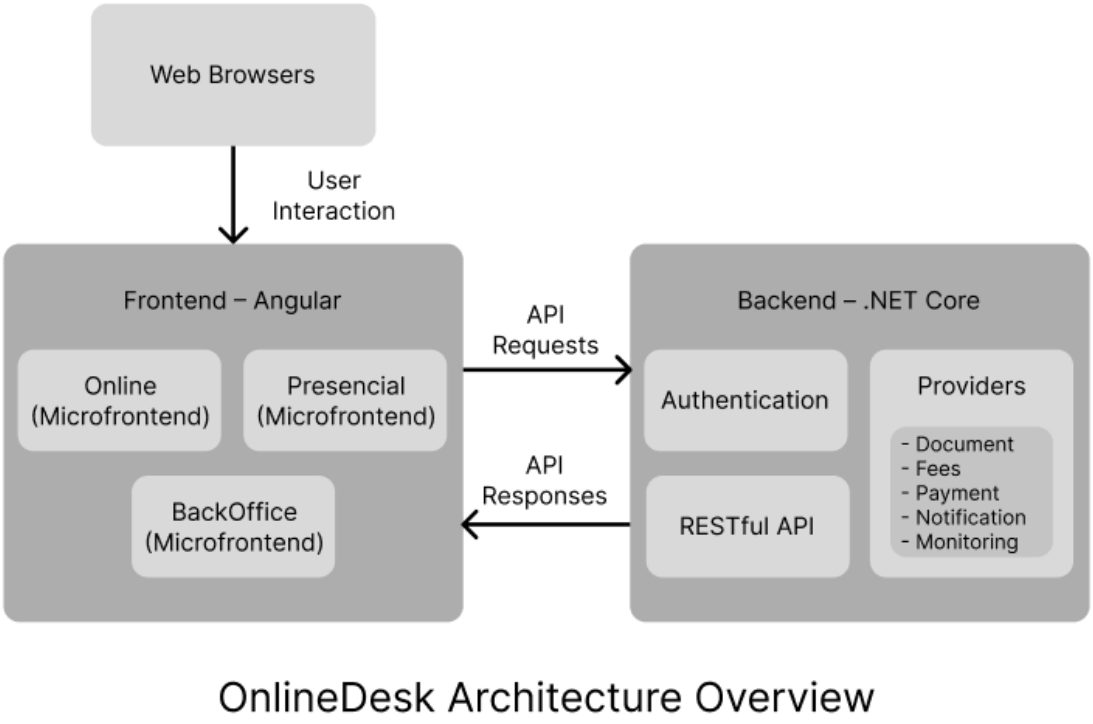


Figure 16 - OnlineDesk Frontend and Backend Architecture

To better understand how information is structured and processed within OnlineDesk, it is important to clarify the key concepts that underpin its data model. At the heart of the system lies the concept of a Request, which represents a submitted instance of a form completed by a user (requester). As shown in Figure 17, each request contains structured information such as submission date, unique identifier, requester's identity, current status, and any supporting documentation. These requests are the central unit of processing and coordination within the platform.

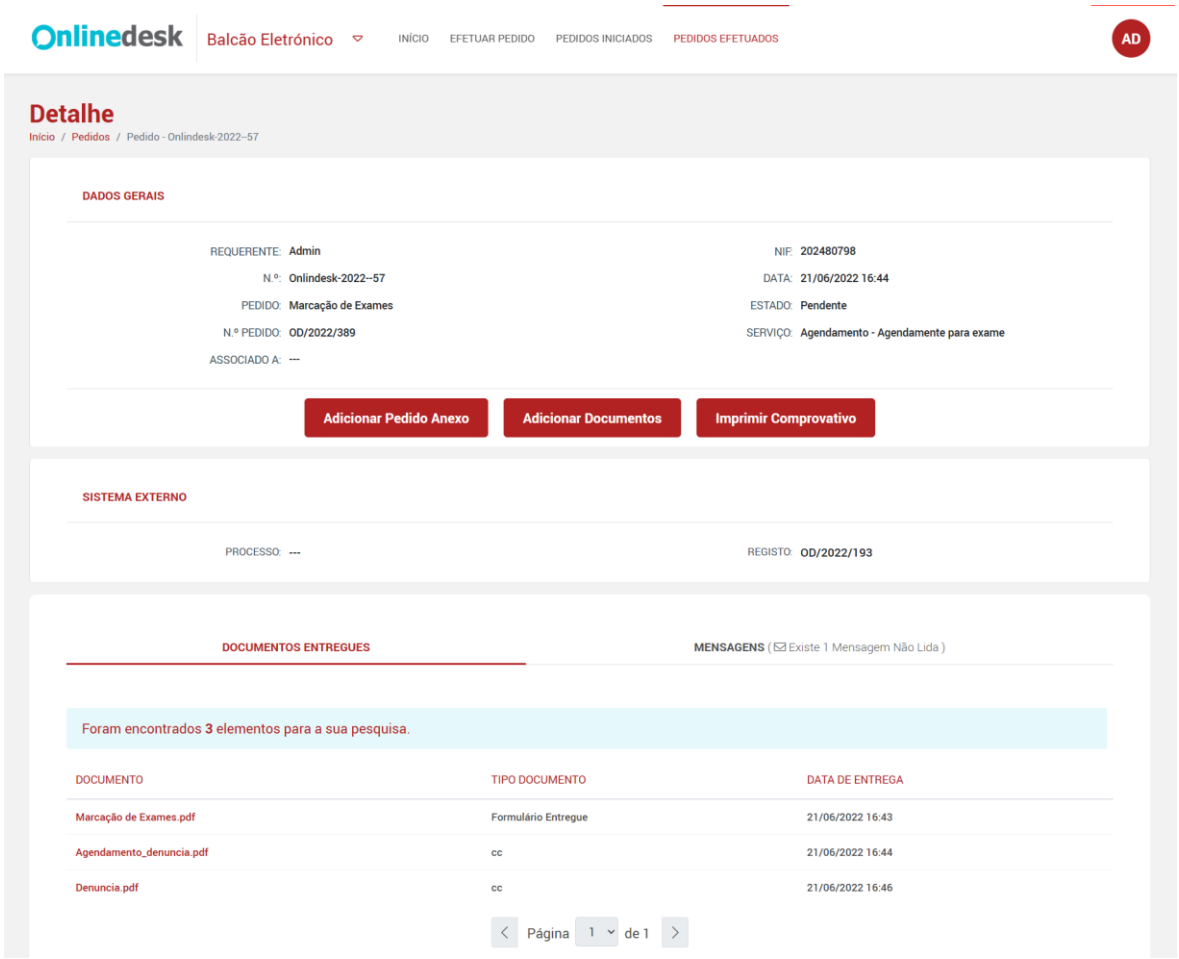


Figure 17 - OnlineDesk Request

Requests are always associated with a Form, which defines the data structure, fields, and documents required for submission. All forms contain fields, validation rules, metadata, document template, optional file attachments and associated services or providers, as illustrated in Figure 18.

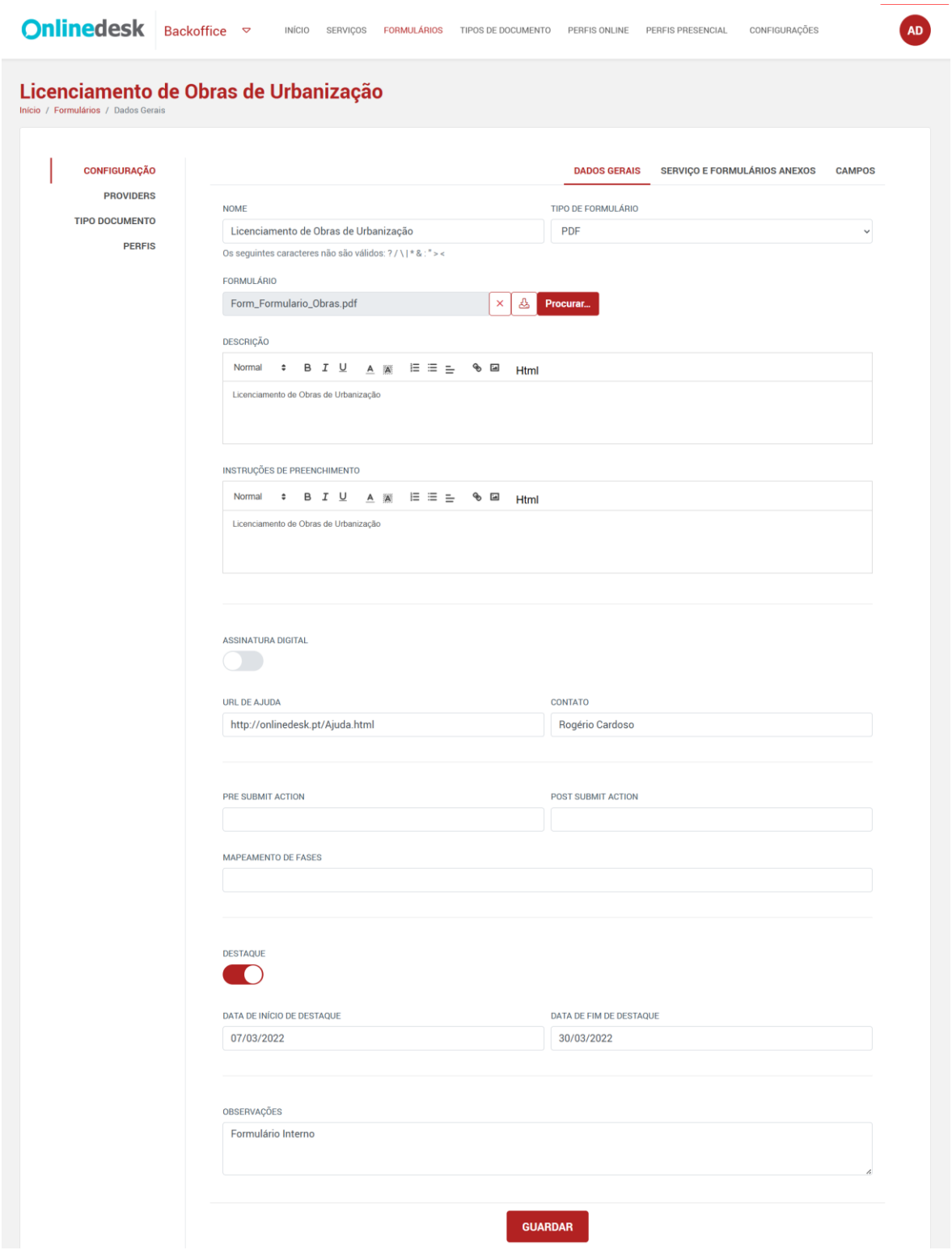


Figure 18 - OnlineDesk Form

Each form is also associated with a Service, which acts as a hierarchical categorization mechanism that helps organize offerings into service and subservice structures. This is exemplified in Figure 19, where a parent service (e.g. “Urbanismo - Licenciamento”) contains subservices (e.g. "Obras de Edificação" and "Obras de Remodelação") with

different associated forms available for user selection (e.g. “Licença de obras de edificação”).

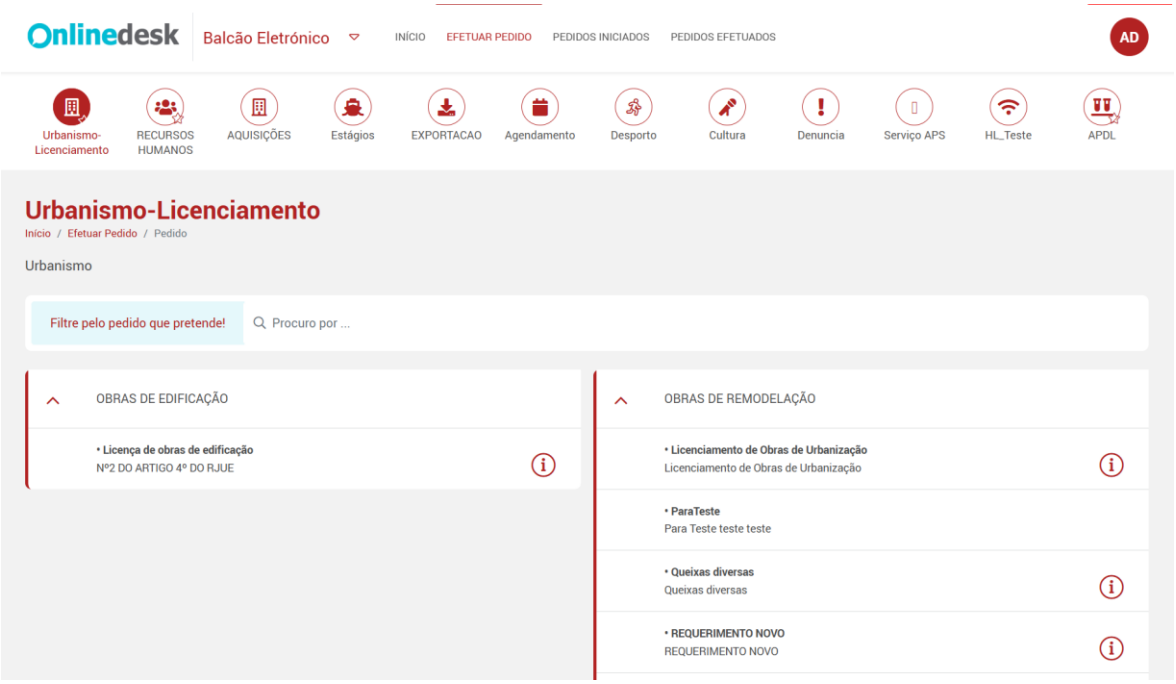


Figure 19 - OnlineDesk Services

Every form may specify one or more Providers, which are responsible for executing domain-specific logic or integration tasks. For example, a document provider may send the submitted request to Edoclink for archival and workflow execution, a fees provider may calculate the applicable fees during submission, a payment provider integrates with payment gateways to define and validate payment methods, a notification provider may send real-time alerts or confirmations to users or internal departments or a monitoring provider may emit lifecycle events for tracking execution status and SLA compliance. This modular design allows OnlineDesk to delegate operations to isolated components while keeping the core system loosely coupled.

Once a form is configured and associated with a service and subservice, it becomes accessible through OnlineDesk’s available user channels (e.g Online or *Presencial*). After completing the form and attaching any required files, the user proceeds with the submission. At that moment, OnlineDesk generates a new Request, which is then persisted and initiates the execution of any configured providers, each responsible for executing specific actions.

The relationships between Requests, Forms, Services, and Providers are summarized in Figure 20.

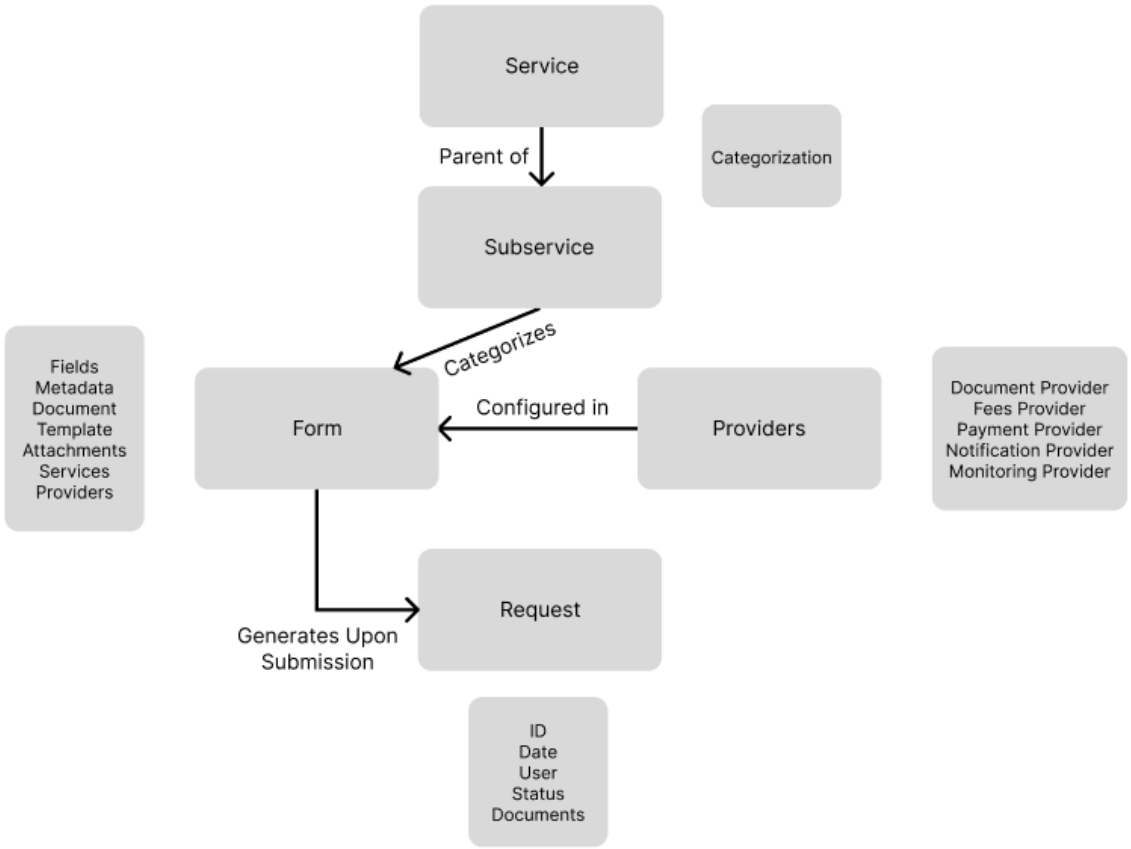


Figure 20 - OnlineDesk Key Concepts Relationships

6.4.Technologies

The architectural design of OnlineDesk and the broader eProcess framework relies on a selected stack of technologies to meet the project's functional and non-functional requirements. These technologies were selected by Link Consulting for their compatibility, scalability, and ability to deliver an efficient user experience. From transitioning legacy SOAP services to REST APIs, to leveraging Angular for dynamic frontend interfaces and .NET for robust backend operations, the chosen tools and frameworks enable the system to operate with high performance and adaptability.

This section outlines the key technologies employed in OnlineDesk, with a focus on how each contributes to achieving the architectural goals, such as modernizing communication protocols, enhancing user interactions, and ensuring system reliability.

6.4.1. SOAP vs. REST: Refactoring Communication Protocols

SOAP is a protocol used in OnlineDesk's legacy architecture, facilitating communication between backend systems through structured data exchange. It is a highly standardized protocol that offers built-in features such as message security (WS-Security), reliable messaging, and transactional support, making it suitable for complex enterprise environments.

Despite its strengths, SOAP also presents significant challenges. As illustrated in Figure 21, its reliance on eXtensible Markup Language (XML) for message formatting leads to larger payloads, increasing processing demands and affecting performance. This verbosity, combined with the protocol's detailed specifications, contributes to significant bandwidth and performance overhead [38]. Additionally, SOAP's strict adherence to predefined Web Services Description Language (WSDL) definitions limits flexibility, making integration with modern systems more challenging.

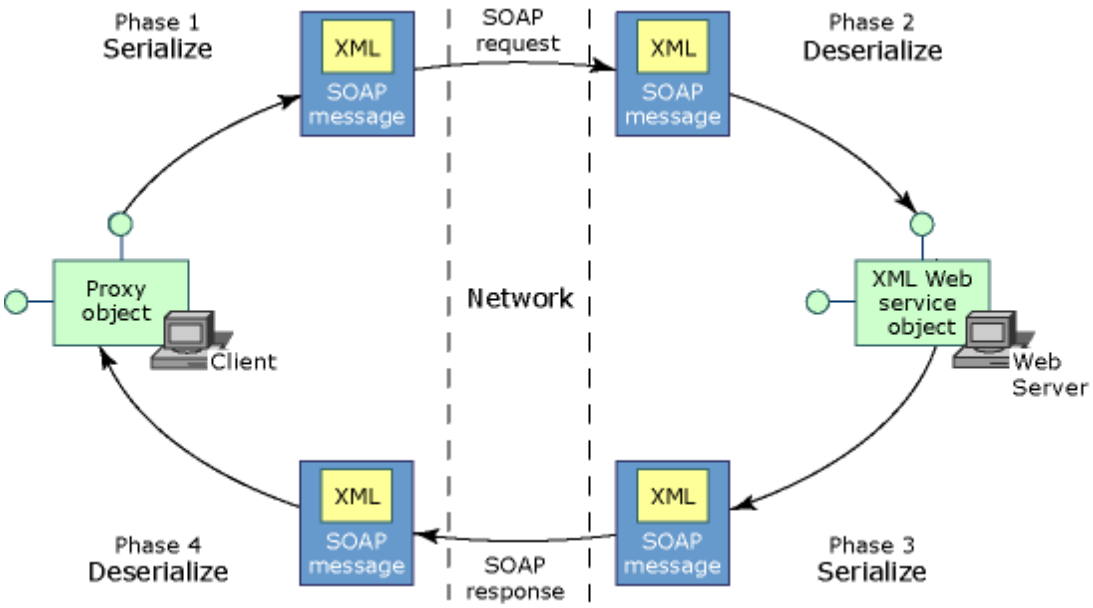


Figure 21 - SOAP Dataflow [39]

While SOAP remains valuable in high-security or enterprise environments with strict requirements, these drawbacks prompted OnlineDesk to adopt REST APIs as the primary communication protocol for its integration with Edoclink. REST offers a simpler, lightweight, and scalable alternative that aligns with modern architectural practices.

REST adopts a stateless architecture, leveraging standard Hypertext Transfer Protocol (HTTP) methods (GET, POST, PUT, DELETE) to perform CRUD (Create, Read, Update, Delete) operations, as seen in Figure 22. This transition offered several key advantages to

OnlineDesk's infrastructure. The simplicity of REST lies in its use of lightweight data formats such as JSON, which reduce message size and simplify parsing and processing. This efficiency translates to improved performance, as reduced overhead and faster message transmission make REST ideal for applications with high traffic [40]. Additionally, REST's resource-oriented design provides the flexibility to adapt quickly to changes in data structures or functionality, ensuring scalability and future-proofing the system [41]. Moreover, REST APIs enjoy wide adoption across programming languages, tools, and platforms, enabling seamless integration with third-party services and enhancing interoperability within the ecosystem.

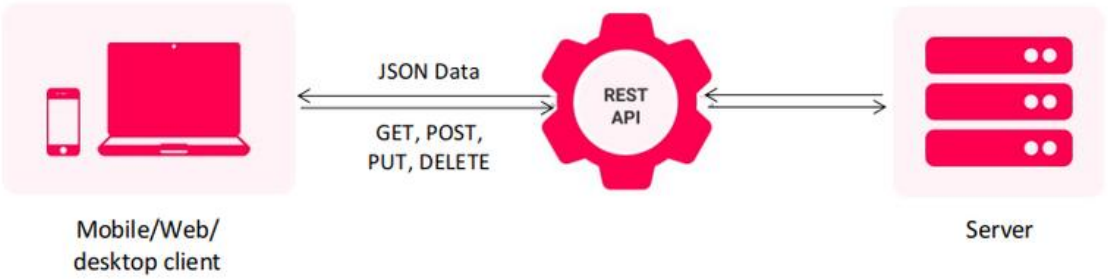


Figure 22 - Rest API Model [42]

The shift from SOAP to REST in OnlineDesk addressed several challenges and introduced modern architectural benefits. By minimizing bandwidth usage and reducing processing requirements, REST enhanced performance, delivering a more responsive and efficient user experience. Its lightweight and stateless design also improved scalability, making it an ideal fit for a modular, microservices-based architecture [43]. REST's simplicity proved advantageous for developers, streamlining both development and maintenance processes compared to the complexities of SOAP.

6.4.2. Angular: Building Dynamic Frontend Interfaces

Angular, a popular JavaScript framework maintained by Google, serves as the foundation for OnlineDesk's frontend development. Renowned for its robust architecture and modular design, Angular empowers developers to create highly dynamic, interactive, and responsive user interfaces that meet the demands of modern web applications.

The features of Angular include its component-based architecture, two-way data binding, built-in features, performance optimizations, and scalability, as shown in Figure 23

[44] [45]. These features make it particularly suited for enterprise-grade applications like OnlineDesk.

Figure 23 - Angular Architecture [46]

Angular employs a component-based architecture, which divides an application into reusable, self-contained units. This design promotes consistency across the platform, accelerates development by enabling code reuse, and reduces redundancy. Each component encapsulates its unique logic and user interface, simplifying the development, testing, and maintenance of features [44] [45].

Angular's two-way data binding further enhances its capabilities by synchronizing the UI with the underlying data model within the frontend. Any changes made in the UI are instantly reflected in the application's underlying data model and vice versa. This synchronization ensures an up-to-date user experience [44] [45].

Additionally, Angular offers a rich array of built-in features that streamline development. Dependency injection simplifies communication between components and services, enabling efficient resource sharing. Angular's routing system ensures smooth navigation between views and modules, which is particularly valuable in multi-page applications like OnlineDesk. Its comprehensive testing tools support both unit and end-to-end testing, ensuring reliability and reducing bugs [44] [45]. For managing complex application states, Angular's ecosystem integrates tools like NgRx, which is essential for maintaining interactivity in service portals.

Performance optimization is another cornerstone of Angular. With Ahead-of-Time (AOT) compilation, applications are compiled during build time rather than runtime, resulting in faster execution [45]. The framework also supports lazy loading, ensuring that only the components required at a given time are loaded, thereby improving responsiveness and resource efficiency.

Angular's modularity makes it an ideal choice for scalable projects like OnlineDesk. Developers can organize the application into feature-specific modules, which not only support scalability as new features are introduced but also simplify collaboration by allowing teams to work on isolated parts of the codebase.

Beyond its technical capabilities, Angular boasts a robust ecosystem of tools, libraries, and third-party integrations, ensuring extensive support and the ability to extend functionality as needed. Its active developer community drives continuous improvement, providing stability and support for long-term projects.

The decision to use Angular for OnlineDesk aligns perfectly with the platform's requirements for complex, customizable interfaces. Angular's form validation, data handling, and dynamic update capabilities are integral to the customizable forms and dashboards that power OnlineDesk's service portals. Furthermore, its performance optimization ensures a responsive and reliable experience for users, even as the platform scales to accommodate growing demands.

6.4.3. .NET: Ensuring Backend Reliability and Scalability

The backend of OnlineDesk is powered by .NET, a versatile and high-performance framework developed by Microsoft. This robust framework serves as the backbone of OnlineDesk's operations, offering the security, scalability, and integration necessary to meet the platform's requirements. Built primarily using the C# programming language, .NET leverages C#'s strong typing, object-oriented capabilities, and modern features to enhance productivity and maintainability while delivering enterprise-grade applications [47].

.NET is an open-source, cross-platform development framework designed to support a wide variety of application types, including web, mobile, and cloud-based solutions [48]. It consists of a unified runtime, comprehensive libraries, and robust tools that streamline the development of scalable systems, as illustrated in Figure 24.

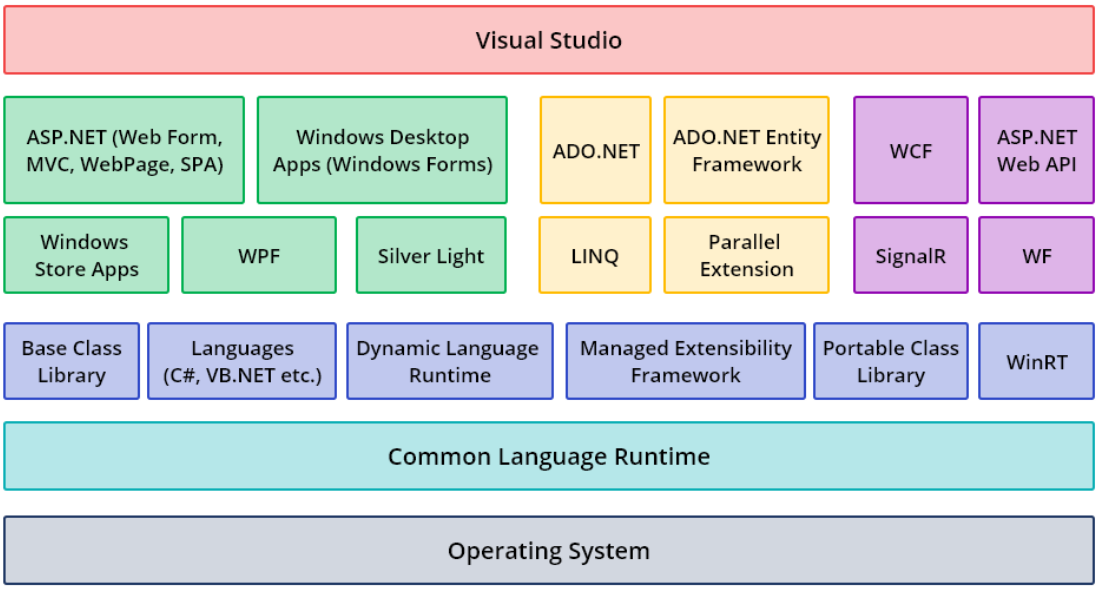


Figure 24 - .NET Framework Architecture [49]

.NET's comprehensive suite of tools, libraries, and frameworks further enhances its appeal. For authentication and authorization, features like ASP.NET Identity and built-in support for OAuth and OpenID Connect streamline secure user management. Data access is simplified with Entity Framework Core, which provides an efficient bridge between the application and its databases [50]. Additionally, ASP.NET Core facilitates the creation of REST APIs, enabling seamless communication between OnlineDesk and other components of the eProcess framework.

The framework's architecture is optimized for high scalability and performance, making it well-suited for OnlineDesk's user base. Asynchronous programming with C# async/await and efficient garbage collection mechanisms ensure optimal resource utilization, while .NET Core's modular design supports both vertical and horizontal scaling to meet increasing demands [51]. These capabilities allow OnlineDesk to maintain high performance, even under significant traffic loads.

Security is another key element of .NET, with its robust mechanisms for safeguarding data and applications. Built-in libraries for encryption, token management, and secure data transmission ensure compliance with industry standards, such as GDPR. Furthermore, integration with secure identity providers like Keycloak enhances the platform's authentication processes, reinforcing its overall security infrastructure [47].

.NET's compatibility with a wide range of communication protocols, including REST, gRPC Remote Procedure Call (gRPC), and WebSockets, emphasizes its ability to facilitate efficient communication across diverse systems. By adhering to standard data formats like JSON and XML, .NET ensures smooth interoperability across diverse systems, enabling OnlineDesk to function cohesively within its ecosystem.

The framework also enhances developer productivity through its rich ecosystem of tools, including Visual Studio and JetBrains Rider. Combined with C# readability and developer-friendly features, these tools streamline the development lifecycle with features such as integrated debugging, profiling, and code analysis, ensuring that high-quality code is produced efficiently.

The decision to use .NET for OnlineDesk was guided by its ability to provide a secure, scalable, and maintainable backend infrastructure. Its tools for API development, data handling, and authentication enable the platform to manage complex workflows and accommodate a growing user base [48] [51]. As OnlineDesk continues to evolve, .NET's modularity and scalability will provide a robust foundation for future enhancements, ensuring that the system remains reliable, adaptable, and capable of meeting the needs of modern organizations.

6.4.4. Middleware and Security

OnlineDesk provides users with multiple authentication methods to ensure secure, flexible, and user-friendly access to its features and data. These methods cater to diverse organizational needs, enabling seamless integration with existing security protocols while maintaining robust access control. The main authentication methods supported by OnlineDesk are:

- Keycloak (OAuth 2.0 Authentication):

  Keycloak, an open-source identity and access management solution, is used to provide OAuth-based authentication. By issuing secure tokens, Keycloak enables Single Sign-On (SSO) capabilities and centralized user management. This approach is particularly suitable for environments requiring scalable and modern security infrastructures, ensuring seamless communication between OnlineDesk and other systems [29] [30].

- Windows Authentication:

  For organizations operating within a Windows ecosystem, OnlineDesk supports Windows Authentication, allowing users to log in with their existing Windows credentials. This integration with Active Directory simplifies access for employees already logged into corporate networks, streamlining the user experience while maintaining enterprise-grade security [31].

- Local Authentication:

  OnlineDesk also offers traditional username and password-based authentication. This option is ideal for standalone environments or scenarios where integration with external identity providers is unnecessary. It allows users to register, manage credentials, and securely log in directly within the platform, without reliance on third-party systems [32].

These authentication methods are powered by middleware components that enhance security by enforcing access control policies and ensuring that only authorized users can interact with OnlineDesk. As illustrated in Figure 25, the login interface provides users with the flexibility to choose their preferred authentication method. This approach ensures OnlineDesk is adaptable to varying security requirements, offering a secure experience aligned with industry-standard authentication practices.

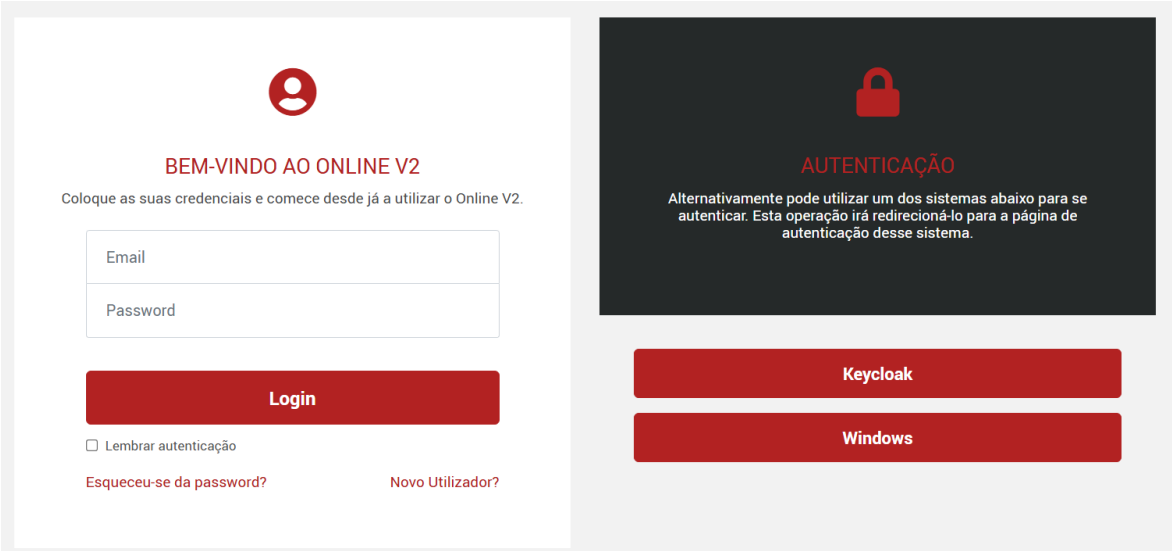


Figure 25 - OnlineDesk Login Interface

6.5.Summary

This chapter has explored the architectural foundation of the eProcess framework, highlighting its modular composition, layered organization, and the technologies selected to support scalability, extensibility, and maintainability. The high-level architecture divides responsibilities between process definition and execution, enabling a flexible design that accommodates evolving business needs.

Each core component was examined in detail, including OnlineDesk, Edoclink, Form Builder, Process Monitor, Process Designer, and eProcess Factory, showcasing their roles and how they integrate to form a cohesive and interoperable BPM platform. Additionally, a dedicated section focused on the internal architecture of OnlineDesk, clarifying key concepts and illustrating how these elements work together to structure data and coordinate workflows. The chapter also reviewed the technological choices underpinning the system, from the adoption of RESTful APIs and secure authentication strategies to the use of Angular for frontend development and .NET for backend logic, emphasizing how these decisions align with modern architectural best practices.

This provides a starting point for the next chapter, which focuses on how these architectural principles were put into practice through the implementation of new features and integrations.

7. Implementation

This chapter presents the technical implementation of the core improvements made to OnlineDesk to enhance its performance, modularity, and integration capabilities within the broader eProcess framework. These enhancements were developed as part of the company project in which this work was integrated, addressing key requirements, including modernization of communication protocols, dynamic form management, event-driven monitoring, and standardized configuration portability.

The chapter begins in section 7.1 (Refactoring SOAP to REST API), detailing the transition from SOAP-based services to REST APIs for OnlineDesk's integration with Edoclink. This migration modernized the communication model, simplified maintenance, and improved performance across key service endpoints. Section 7.2 (Integration with the Form Builder) describes how the platform was extended to dynamically retrieve, render, and process forms from an external system, providing users with greater flexibility in form design and submission workflows. In section 7.3 (Integration with the Process Monitor), the focus shifts to enhancing observability through the implementation of an event-based integration with the Process Monitor using Kafka for real-time SLA tracking and process transparency. Finally, section 7.4 (Importing Services and Forms via JSON) introduces a mechanism for importing service and form configurations from structured JSON files, enabling standardized and automated configuration replication across environments and eProcess components.

Together, these implementations improve OnlineDesk's extensibility and interoperability, while aligning the platform with modern architecture practices and supporting future scalability. The following sections detail the technical reasoning, implementation strategies, and challenges encountered for each improvement.

7.1. Refactoring SOAP to REST API

The refactoring of OnlineDesk's integration with Edoclink involved replacing SOAP-based services with REST APIs to modernize communication and improve system performance. This transition aimed to enhance scalability, simplify maintenance, and align OnlineDesk with contemporary API standards.

The following sections explore the motivations behind this transition, the migration strategy, the technical refactoring process, challenges encountered, and the benefits achieved.

7.1.1. Purpose of the Refactoring

The transition from SOAP-based services to REST API was essential for modernizing OnlineDesk's communication with Edoclink, as the existing SOAP implementation introduced significant performance, scalability, and maintainability challenges [38]. This migration reduced overhead by replacing SOAP's verbose XML messaging with REST's lightweight, resource-oriented architecture, improving both network efficiency and maintainability. Additionally, the shift enhanced scalability due to REST's stateless nature, which allowed for easier horizontal scaling and made it better suited for future growth [41]. Moreover, the transition helped standardize API communication by aligning OnlineDesk with modern industry practices, ultimately improving integration with other eProcess components.

By moving to REST, the system could take advantage of standardized HTTP methods such as GET, POST, PUT, and DELETE, enabling a more intuitive way to interact with resources. Furthermore, REST's widespread adoption in web services ensured easier integration with frontend applications and external systems, reducing development complexity while enhancing interoperability. With these improvements in mind, a structured migration strategy was designed to ensure a seamless transition from SOAP to REST while maintaining feature parity across the system.

This transition is visually summarized in Figure 26, which contrasts the SOAP-based communication model previously used with the new RESTful approach, highlighting the differences in communication flow, message formatting, and authentication mechanisms.

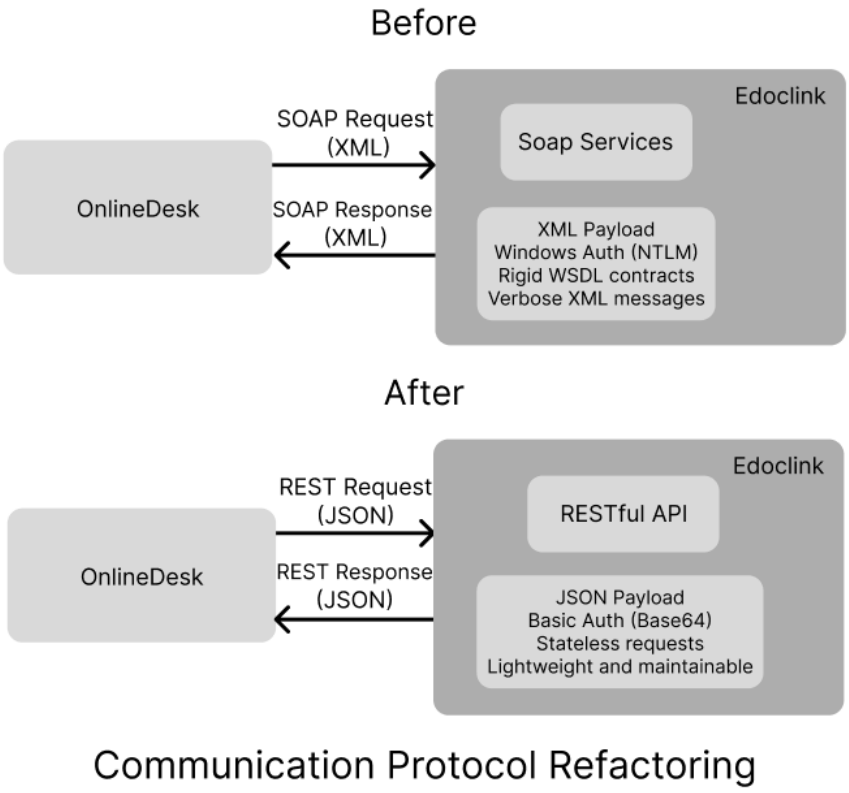


Figure 26 - Transition from SOAP-based to REST API between OnlineDesk and Edoclink

7.1.2. Mapping SOAP Methods to REST Endpoints

The migration process began with an analysis of the existing SOAP services implemented in the existing service class (EdocSVC), which served as the bridge between OnlineDesk and Edoclink. This process involved identifying all SOAP methods present in EdocSVC, mapping each method to its corresponding endpoint in the Edoclink REST API, by documenting all the forty-eight identified SOAP services that required refactoring [43]. Figure 27 exemplifies a subset of this mapping, showcasing how specific SOAP-based methods were transitioned to REST endpoints, while the complete mapping is provided in Annex 1 for reference. Each row in the table represents a specific SOAP method, its corresponding REST API endpoint, and the HTTP method used in the new implementation. By conducting this analysis, the transition was structured to maintain feature parity while leveraging the advantages of the REST architecture.

Método Provider	Servico Edoc v7	Método Edoc v7	API edoc v8	Operaç	Nota
AdicionarRegisto	Cards	Add2Request	/publicapi/documents	POST	
UpdateRegisto	Cards	UpdateRequest	/publicapi/documents/{id}	PATCH	
AdicionaEntidadeAoRegisto	ServicesSimple	CardEntitiesAddRequest	/publicapi/documents/{documentId}/entities/{id}	POST	
AdicionaDocumentosAoRegistoPorN	ServicesSimple	CardDocumentsAddFullByNameRe	/publicapi/documents/{documentId}/files	POST	
SetReadyRegisto	ServicesSimple	CardSetReadyRequest	/publicapi/documents/{id}	PATCH	
GetDocumentBytes	ServicesSimple	CardDocumentVisualizationURLRe	/publicapi/files/{id}/content	GET	
AdicionaEntidadeExternaAoRegisto	ServicesSimple	CardEntityAddRequest	/publicapi/documents/{documentId}/entities/{id}	POST	
GetEdocDocumentTypes	DocumentTypes	ListRequest	/publicapi/filetypes	GET	
GetEdocDocumentTypeByInternalBo	DocumentTypes	ListAsXML2Request	/publicapi/documentTypes/{documentTypeId}/fileTypes	GET	
GetEdocCardDocumentsCount	Cards	DocumentsListRequest	/publicapi/documents/{documentId}/files	GET	
GetEdocCardDistributionsList	Cards	DistributionsListRequest	/publicapi/documents/{documentId}/flows	GET	
GetEdocCard	ServicesSimple	CardLoadRequest	/publicapi/documents/{id}	GET	

Figure 27 - SOAP to REST Mapping

For example, the "AdicionarRegisto" method, which was originally handled in SOAP as Add2Request under the Cards service, was mapped to the REST endpoint /publicapi/documents using the POST operation. This change ensures that new document records are created using a more streamlined, lightweight REST request instead of a SOAP-based XML payload. Similarly, operations such as retrieving document details, like GetEdocCard, transitioned from CardLoadRequest in SOAP to a GET request at /publicapi/documents/{id}, simplifying data retrieval with a direct HTTP call.

7.1.3. Refactoring Process

With a clear mapping of SOAP services to their REST counterparts, the next phase focused on refactoring each service while ensuring that all business logic remained intact. This process involved replacing XML-based SOAP requests with more efficient JSON-based REST interactions.

To illustrate this transition, Figure 28 and Figure 29 provide an example of how one of the forty-eight refactored methods was transformed. Figure 28 presents the original SOAP-based implementation, which relies on XML message formatting and service proxies for communication with the backend. The method begins by creating a communication channel using ChannelFactory (line 6-9), which is necessary for invoking SOAP services. The GetChannel method is responsible for setting up this communication, selecting the appropriate service endpoint, and configuring authentication using Windows Authentication (NTLM). The credentials are passed explicitly (domain, username, and password), and the connection is established through a BasicHttpBinding, which supports SOAP-based communication over HTTP. A request object (ListAsXmlRequest) is then created and passed to the service (line 12). The response is received as an XML string, which must be parsed

and deserialized into a dataset (DataSet dsBooks) (line 16-19). Finally, the method iterates over the dataset to extract and return structured data as a list of Book objects (line 21-28).

```
1  public static List<Book> GetEdocBooksList(string ID, string credentials,
                                              IProviderServiceFactory xfactory)
2  {
3      List<Book> listBooks = new List<Book>();
4      DataSet dsBooks = new DataSet();
5
6      var factory = (ChannelFactory<eDocBooksSVC.Books>)GetChannel("books",
                     credentials);
7      var serviceClient = xfactory.CreateChannel(factory);
8      ((ICommunicationObject)serviceClient).Open();
9
10     try
11     {
12         var WSreq = new eDocBooksSVC.ListAsXmlRequest(1, -1, null,
                       eDocBooksSVC.BookListMode.CanWriteCards);
13         AuditRequest(ID, MethodBase.GetCurrentMethod().Name, WSreq);
14         DateTime dateStart = DateTime.Now;
15         AppContext.SetSwitch("System.Net.Http.UseSocketsHttpHandler", false);
16         string xml =
                   serviceClient.ListAsXmlAsync(WSreq).Result.ListAsXmlResult;
17         AuditExecutionTime(ID, MethodBase.GetCurrentMethod().Name, dateStart,
                              WSreq);
18
19         dsBooks.ReadXml(new StringReader(xml));
20
21         for (int i = dsBooks.Tables[0].Rows.Count - 1; i > -1; i--)
22         {
23             Book livro = new Book
24             {
25                 Name = dsBooks.Tables[0].Rows[i]["name"].ToString()
26             };
27             listBooks.Add(livro);
28         }
29     }
30     catch (Exception ex)
31     {
32         throw;
33     }
34     finally
35     {
36         factory.Close();
37         ((ICommunicationObject)serviceClient).Close();
38     }
39     return listBooks;
40 }
```

Figure 28 - GetEdocBooksList() SOAP-Based Method

In contrast, Figure 29 displays the equivalent REST-based implementation, which simplifies communication by using standard HTTP requests. Rather than constructing and passing a SOAP request object, this implementation makes an HTTP GET request directly

to the REST API endpoint (line 12-13). Upon receiving a response, the method extracts the JSON payload and deserializes it into a list of Book objects using a JSON parsing library (line 20-27). Unlike the SOAP implementation, which required manually constructing a dataset from XML, this approach allows for direct object mapping, significantly reducing overhead.

```
1  public async Task<List<BookV8>> GetEdocBooksList(string ID, int? page =
                                    null, int? pagesize = null, bool? count =
                                   null, string orderby = null, string filter
                                  = null)
2  {
3      LogHelper lg = new LogHelper();
4      var queryParams = new List<string>();
5      if (page.HasValue) queryParams.Add($"page={page}");
6      if (pagesize.HasValue) queryParams.Add($"pagesize={pagesize}");
7      if (count.HasValue) queryParams.Add($"count=
                                        {count.ToString().ToLower()}");
8      if (!string.IsNullOrEmpty(orderby)) queryParams.Add($"orderby=
                                                    {orderby}");
9      if (!string.IsNullOrEmpty(filter)) queryParams.Add($"filter={filter}");
10
11     var queryString = queryParams.Count > 0 ? "?" + string.Join("&",
                         queryParams) : string.Empty;
12     var fullUrl = $"/publicapi/documentTypes{queryString}";
13     var response = await _client.GetAsync(fullUrl);
14
15     if (!response.IsSuccessStatusCode)
16     {
17         throw new Exception($"API Error: {response.StatusCode} -
                              {response.ReasonPhrase}");
18     }
19
20     var responseBody = await response.Content.ReadAsStringAsync();
21
22     var options = new JsonSerializerOptions { PropertyNameCaseInsensitive =
                                                 true };
23     var documentTypesResponse =
          System.Text.Json.JsonSerializer.Deserialize<DocumentTypesResponse>
          (responseBody, options);
24
25     var books = documentTypesResponse.Result.Select(documentType => new
                 BookV8 { Name = documentType.Key.Name,
26                        ID = documentType.Key.ID })
27                 .ToList();
28
29     return books;
30 }
```

Figure 29 - GetEdocBooksList() REST-Based Method

7.1.4. Updating System Dependencies

Once the refactoring of the EdocSVC class was completed, further modifications were required in the components that interacted with it. The primary dependencies affected by this transition were the EdocEndpointManager and EdocEntityManager classes. These components were originally designed to invoke the SOAP-based methods, meaning they needed to be updated to call the newly implemented REST endpoints in EdocSVC instead of the deprecated SOAP services. The changes included updating function calls, modifying request and response handling, and adapting error management mechanisms to align with REST API standards.

Figure 30 and Figure 31 showcase how methods in EdocEndpointManager and EdocEntityManager were updated to call the new REST APIs, ensuring full integration across the system. These examples demonstrate the structural differences between the old and new implementations.

Figure 30 presents the original SOAP-based implementation, where the GetBooksList method calls the SOAP service through EdocSVC.GetEdocBooksList, passing the ID, credentials, and factory instance (line 7). The method waits for a response from the SOAP service and processes the returned data synchronously.

```
1  public List<Book> GetBooksList()
2  {
3      string ID = "_" + String.Format("{0:0000}", new Random().Next(1, 9999));
4
5      List<Book> saida = new List<Book>();
6
7      saida = EdocSVC.EdocSVC.GetEdocBooksList(ID, eDocCredentials, factory);
8
9      return saida;
10 }
```

Figure 30 - GetBooksList() SOAP-Based Method Invocation

In contrast, Figure 31 illustrates the REST-based implementation, where GetBooksList has been refactored to interact with the REST API. One notable change is the removal of the factory parameter, as REST does not require a dedicated service factory to establish communication channels like SOAP. Additionally, while SOAP-based methods were previously invoked synchronously due to the structure of the generated WCF service proxies, the transition to REST using HttpClient allowed the service to adopt asynchronous programming patterns. In this context, the refactored method makes use of

.GetAwaiter().GetResult() to explicitly manage the asynchronous REST call and retrieve the API response reliably (line 10).

```
1 public List<BookV8> GetBooksList()
2 {
3     this.LoadProviderConfiguration();
4     this.SetupProvider();
5
6     string ID = "_" + String.Format("{0:0000}", new Random().Next(1, 9999));
7
8     List<BookV8> saida = new List<BookV8>();
9
10    saida = edocSVC.GetEdocBooksList(ID).GetAwaiter().GetResult();
11
12    return saida;
13 }
```

Figure 31 - GetBooksList() REST-Based Method Invocation

7.1.5. Challenges and Solutions

Although the transition from SOAP to REST delivered significant benefits, several challenges emerged throughout the process. One of the primary obstacles involved handling authentication mechanisms that were previously tied to SOAP-specific security features. SOAP relied on NTLM and WS-Security for message encryption and integrity verification, whereas REST operates with more flexible authentication models such as Basic Authentication, which is the mechanism used to communicate with the Edoclink API [40]. This required careful adaptation to ensure that OnlineDesk's API interactions remained secure while leveraging REST authentication mechanisms.

Another key challenge was the need to ensure that the refactored REST-based methods retained exactly the same behavior as their SOAP-based counterparts. Since OnlineDesk depended on these services for business operations, any deviation in functionality could disrupt workflows. Each of the forty-eight methods within the EdocSVC class underwent testing to confirm that the refactored implementation functioned as expected. The testing process involved validating API responses for each method, ensuring that all interactions with Edoclink produced the correct output. Following this, the EdocEndpointManager and EdocEntityManager classes were also tested to verify that their methods successfully interacted with the updated EdocSVC. These classes were previously designed to interact with SOAP-based services, meaning their method calls, response handling, and error

management mechanisms had to be adjusted. Any inconsistencies in how these layers processed requests or responses could lead to functional regressions.

Finally, another challenge stemmed from the differences in message formatting between SOAP and REST. SOAP-based communication heavily relied on XML, enforcing strict schema definitions that dictated how data was structured. The shift to REST introduced JSON as the preferred format, requiring modifications in data parsing and validation processes.

7.1.6. Summary

In summary, the migration from SOAP to a REST API marked a significant improvement in OnlineDesk's communication with Edoclink. By analyzing and refactoring the SOAP services, the system successfully transitioned to a more efficient, scalable, and maintainable API structure. This transformation not only reduced network latency and improved response times but also simplified API management by eliminating the complexities associated with SOAP-based interactions. While challenges such as adapting authentication mechanisms and handling message format changes required resolution, the overall impact of the migration was positive. The transition positioned OnlineDesk for greater interoperability, future scalability, and integration with modern web technologies, ensuring that the platform remains adaptable.

7.2. Integration with the Form Builder

The integration of Form Builder into OnlineDesk was implemented to provide a structured and configurable method for creating and managing forms. Rather than relying on predefined form structures within OnlineDesk, this integration allows forms to be dynamically defined and managed within Form Builder, reducing the need for code modifications when requirements change. When a request is created using a Form Builder template, the user is redirected to the corresponding Form Builder environment. After completing and submitting the form, the structured data and a generated PDF are automatically transmitted back to OnlineDesk for processing and storage. This section details the technical implementation of this integration, covering backend API communication, authentication, frontend enhancements, and plug-in development.

7.2.1. Purpose of the Integration

The primary goal of this integration is to streamline form creation and ensure flexibility in how forms are used within OnlineDesk workflows. Instead of relying on predefined structures, OnlineDesk now supports dynamically managed forms, allowing users to adapt them as needed without modifying the platform's underlying code.

With this new approach, users can create and customize forms directly within the Form Builder, tailoring them to specific processes without the need for backend modifications. This ensures that forms remain adaptable and responsive to evolving workflow requirements.

By maintaining form structures and configurations within the Form Builder, OnlineDesk effectively decouples form management from its internal logic. This centralization not only simplifies maintenance but also enhances scalability. Once a form is submitted, the system automates data handling, validating inputs, storing information, and forwarding the collected data to the relevant systems.

By handling form definitions externally, OnlineDesk achieves a scalable and adaptable form management system that minimizes maintenance overhead while supporting workflow-driven environments.

As shown in Figure 32, the integration between OnlineDesk and Form Builder involves three key interactions: retrieving available templates via authenticated API requests, redirecting users to fill out the selected form, and submitting the completed form back to OnlineDesk for processing and storage.

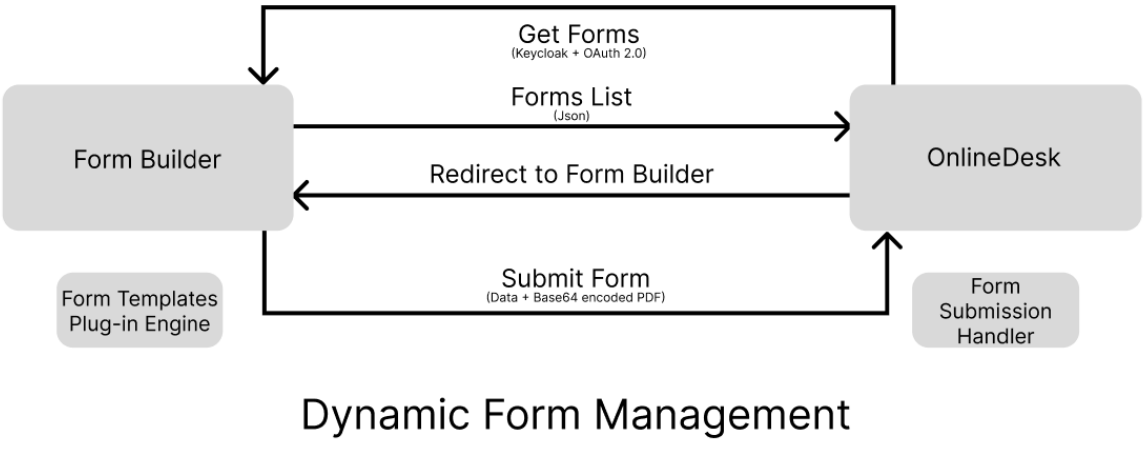


Figure 32 - Integration Flow Between OnlineDesk and Form Builder

7.2.2. Establishing the Configuration for Form Builder Connectivity

To enable the integration between OnlineDesk and Form Builder, the initial step involved configuring the system to store and manage the necessary parameters for API communication. OnlineDesk has a dedicated configuration management system, where various operational parameters are stored and retrieved dynamically. To facilitate the connection with Form Builder, new configuration keys were introduced specifically for managing authentication and API interaction.

These changes were primarily implemented at the backend level, where a new configuration group named "Link Form Builder" was created in the database. This group serves as a structured container for storing multiple configuration parameters required for the integration, including the API base URL for communicating with Form Builder (LinkFormBuilderApiUrl), the client ID (LinkFormBuilderClientId) and client secret (LinkFormBuilderClientSecret) used for authentication, the Keycloak base URL (LinkFormBuilderKeycloakUrl), the realm used for authentication within Keycloak (LinkFormBuilderRealm), and the URL used for rendering forms (LinkFormBuilderRenderUrl). As Form Builder already relies on Keycloak as its identity provider, the integration was implemented to support OAuth 2.0 authentication flows compatible with that setup [29] [30].

These configuration parameters were inserted into the database configuration table and made accessible via the OnlineDesk configuration interface. Figure 33 showcases the newly added configuration section for Form Builder within the OnlineDesk settings.

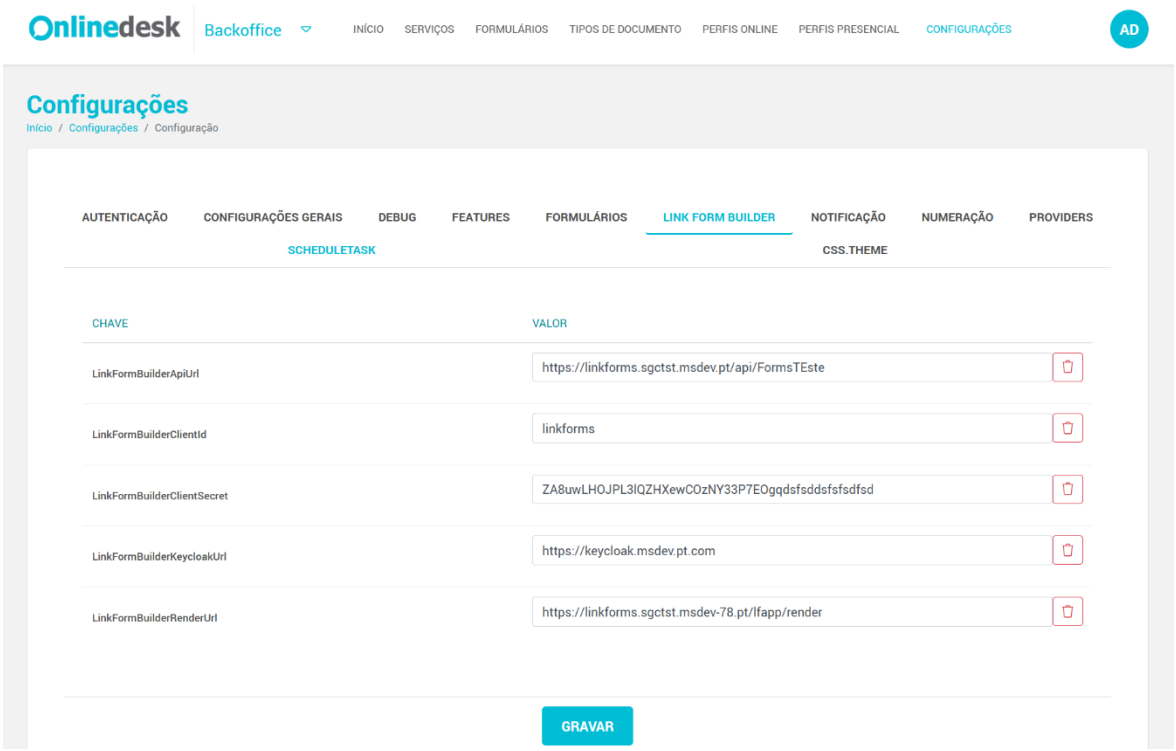


Figure 33 - Form Builder Configurations

By managing these parameters dynamically through the configuration module, the integration remains flexible, allowing administrators to modify API-related settings without requiring direct changes to the application's codebase.

7.2.3. Service for Form Builder API Communication

To establish communication with the Form Builder API, a dedicated service, LinkFormBuilderService, was created in OnlineDesk. This service is responsible for managing authentication with Keycloak to obtain API access tokens, retrieving form data from the Form Builder API, and ensuring configurability, with all relevant API parameters stored in OnlineDesk's configuration system.

The Form Builder API requires authentication through Keycloak, using OAuth 2.0 password grant flow. To facilitate this, the GetAccessTokenAsync() method was implemented to obtain and cache authentication tokens, as illustrated in Figure 34.

```
private async Task<string> GetAccessTokenAsync()
{
    string exception = null;
    var keycloakBaseUrl = _configuration.GetValue<string>
        (ConfiguracoesConstants.GroupLinkFormBuilder,
        ConfiguracoesConstants.LinkFormBuilderKeycloakUrl);
    var realm = _configuration.GetValue<string>
        (ConfiguracoesConstants.GroupLinkFormBuilder,
        ConfiguracoesConstants.LinkFormBuilderRealm);
    var clientId = _configuration.GetValue<string>
        (ConfiguracoesConstants.GroupLinkFormBuilder,
        ConfiguracoesConstants.LinkFormBuilderClientId);
    var clientSecret = _configuration.GetValue<string>
        (ConfiguracoesConstants.GroupLinkFormBuilder,
        ConfiguracoesConstants.LinkFormBuilderClientSecret);

    var keycloakUrl = $"{keycloakBaseUrl}/realms/{realm}/protocol/openid-connect/token";

    if (!string.IsNullOrEmpty(_accessToken) && _tokenExpiration > DateTime.UtcNow)
    {
        return _accessToken; // Use cached token if still valid
    }

    var content = new FormUrlEncodedContent(new[]
    {
        new KeyValuePair<string, string>("client_id", clientId),
        new KeyValuePair<string, string>("grant_type", "client_credentials"),
        new KeyValuePair<string, string>("client_secret", clientSecret)
    });

    try {
        var response = await _httpClient.PostAsync(keycloakUrl, content);
        //response.EnsureSuccessStatusCode();

        if (!response.IsSuccessStatusCode)
        {
            exception = "Falha na autenticação com o Keycloak. Verifique as
                        credenciais.";
            throw new Exception(exception);
        }

        var responseString = await response.Content.ReadAsStringAsync();
        using var jsonDoc = JsonDocument.Parse(responseString);

        if (!jsonDoc.RootElement.TryGetProperty("access_token", out var token))
        {
            exception = "A resposta do Keycloak não continha um token de acesso.";
            throw new Exception(exception);
        }

        _accessToken = jsonDoc.RootElement.GetProperty("access_token").GetString();
        var expiresIn = jsonDoc.RootElement.GetProperty("expires_in").GetInt32();

        _tokenExpiration = DateTime.UtcNow.AddSeconds(expiresIn - 60); // Cache token

        return _accessToken;
    }
    catch (Exception ex)
    {
        if (exception != null)
        {
            throw;
        }
        throw new Exception("Falha na autenticação com o Keycloak. Verifique o URL.");
    }
}
```

Figure 34 - GetAccessTokenAsync() Method

The method begins by retrieving the required authentication credentials, such as the Keycloak URL, realm, client ID, and client secret, from OnlineDesk's configuration system (lines 4-11). These values are stored within the system's configuration database, allowing for easy updates without modifying the code.

Before making an authentication request, the method checks whether a valid access token is already available and has not expired. If the token is still valid, it is reused to reduce the number of authentication requests and minimize delays (lines 15-18). If the token has expired, the method constructs an HTTP request containing the credentials and sends it to the Keycloak authentication server (line 28).

Once Keycloak responds with an access token, the response is parsed, and the token is extracted (lines 37-46). Alongside the token, Keycloak provides an expiration time, which the method uses to calculate when the token will no longer be valid (lines 47-48). The token is then cached to be reused for future API requests until it expires.

Once authentication is established, the service can securely request forms from Form Builder. The GetFormsAsync() method, presented in Figure 35, is responsible for handling this operation.

```
1  public async Task<string> GetFormsAsync()
2  {
3      var apiUrl = _configuration.GetValue<string>
4          (ConfiguracoesConstants.GroupLinkFormBuilder,
           ConfiguracoesConstants.LinkFormBuilderApiUrl);
5
6      // Get Keycloak Access Token
7      string accessToken = await GetAccessTokenAsync();
8
9      // Set Authorization Header
10     _httpClient.DefaultRequestHeaders.Authorization = new
           AuthenticationHeaderValue("Bearer", accessToken);
11
12     // Make API request
13     var response = await _httpClient.GetAsync(apiUrl);
14     //response.EnsureSuccessStatusCode();
15
16     if (!response.IsSuccessStatusCode)
17     {
18         throw new Exception("Erro ao obter formulários.");
19     }
20
21     return await response.Content.ReadAsStringAsync();
22 }
```

Figure 35 - GetFormsAsync() Method

This method first retrieves the API URL from OnlineDesk's configuration system, ensuring that the endpoint remains configurable (line 3). Before sending the request, it ensures that a valid authentication token is available by calling GetAccessTokenAsync() (line 7). This guarantees that all requests to the Form Builder API are properly authenticated.

After obtaining the token, the method sets the Authorization header of the HTTP request using the token value (line 10). This step ensures that the request is recognized as secure and authorized by the Form Builder API.

With authentication in place, the method constructs a GET request to the Form Builder API and sends it (line 13). Upon receiving the response, the method checks whether the request was successful, and if it is, the response is read and returned as a JSON string, which contains the list of available forms (lines 16-21).

7.2.4. API Endpoint in OnlineDesk

To expose the retrieved forms to the frontend, an API endpoint was created within OnlineDesk, mapped to GET /api/FormularioTemplate/linkformBuilder. This endpoint allows the frontend to make HTTP requests to retrieve available forms dynamically.

```
1 [HttpGet("linkformbuilder")]
2 [Authorize(Policy.ApiFormulario)]
3 public async Task<IActionResult> ListForms()
4 {
5     try
6     {
7         var linkFormBuilderService = new LinkFormBuilderService(null, null, _logger,
                                                _configuration);
8         var response = await linkFormBuilderService.GetFormsAsync();
9         var parsedResponse = JArray.Parse(response);
10
11        var formsList = JsonConvert.DeserializeObject<List<LinkFormBuilderPesquisa>>
                        (parsedResponse.ToString());
12
13        var customList = new CustomList<LinkFormBuilderPesquisa>
14        {
15            Items = formsList,
16            ItemsNumber = formsList.Count
17        };
18
19        return OkResult(_mapper.Map<CustomList<LinkFormBuilderPesquisa>>(customList));
20    }
21    catch (Exception ex)
22    {
23        return Invalid(ex.InnerException?.Message ?? ex.Message);
24    }
25 }
```

Figure 36 - LinkFormBuilder API Endpoint

The method ListForms(), as shown in Figure 36, serves as the primary access point for fetching forms. When the frontend makes a request to this endpoint, the method calls GetFormsAsync() (line 8), triggering the process of obtaining an authentication token, contacting the Form Builder API, and retrieving form data.

Once the data is received, the method parses the JSON response, extracting the list of forms from the API's response structure (line 9). The extracted data is then transformed into an object model that is compatible with OnlineDesk's frontend (lines 11-17). This ensures that the data is well-structured and ready to be displayed in the user interface.

Finally, the method returns the formatted list of forms in an HTTP response (line 19), allowing the frontend to render them dynamically.

7.2.5. Frontend Modifications

After establishing a backend connection to Form Builder, the next step involved modifying the OnlineDesk frontend to retrieve and display the available form templates dynamically. This required introducing new data structures, updating the service layer, and modifying the UI components responsible for handling form selection.

Figure 37 illustrates the updated form selection interface in OnlineDesk, where users can now choose the Form Builder type form and then select from the available templates retrieved from Form Builder. The dropdown dynamically loads these templates, ensuring that the latest options are always presented.

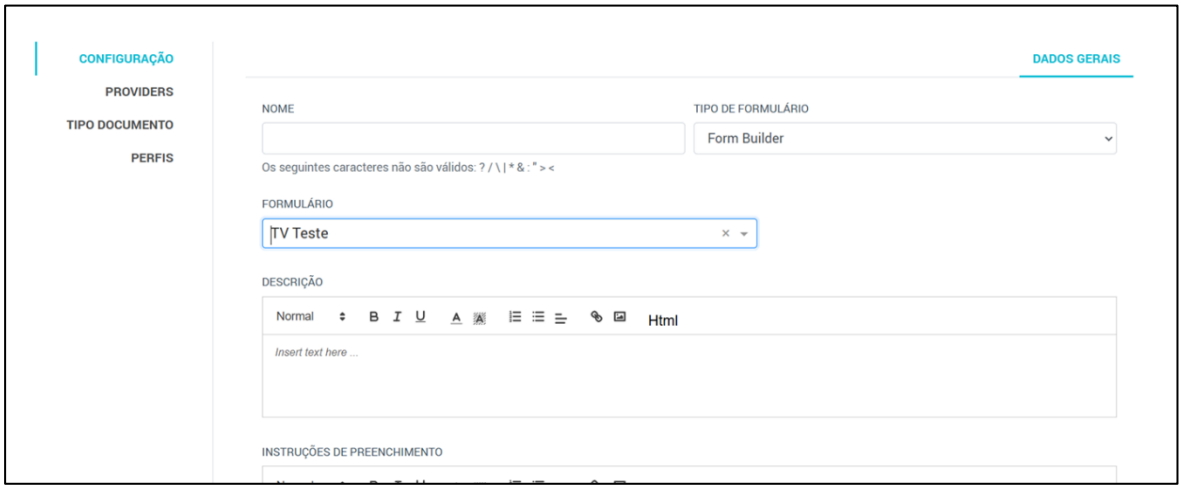


Figure 37 - Form Creation

7.2.6. Handling Form Builder Templates in OnlineDesk

To ensure seamless integration with Form Builder, modifications were made to how OnlineDesk handles the storage, retrieval, and redirection of forms linked to Form Builder. Instead of storing only the form ID, OnlineDesk appends the pre-configured LinkFormBuilderRenderUrl (a base URL pointing to Form Builder's rendering interface) along with the form ID retrieved from Form Builder. This allows for direct redirection to the form when accessed.

When saving a form, OnlineDesk automatically formats the URL before storing it in the database. As shown in Figure 38, in the endpoint mapped to POST /api/formularios (Add method), before persisting the form, the system checks if it is a Form Builder template (TipoFormulario.FORMBUILDER) (line 15). If so, the Ficheiro.Url field is updated to include the Form Builder rendering URL with the form ID as a query parameter (line 18).

```
1  [HttpPost]
2  [AuthorizeAny(Policy.ApiBo, Policy.ApiFormulario)]
3  public IActionResult Add(FormularioDadosGeraisRequest data)
4  {
5      // model validation
6      if (!ModelState.IsValid)
7      {
8          return Invalid(ModelState);
9      }
10
11     try
12     {
13         var formulario = _mapper.Map<Formulario>(data);
14
15         if (formulario.TipoFormularioId ==
               (int)Common.Constants.TipoFormulario.FORMBUILDER)
16         {
17             string formBuilderUrlPrefix = _configuration.GetValue<string>
                                              (ConfiguracoesConstants.GroupLinkFormBuilder,
                                             ConfiguracoesConstants.LinkFormBuilderRenderUrl);
18             formulario.Ficheiro.Url = $"{formBuilderUrlPrefix}?id=
                                              {formulario.Ficheiro.Url}";
19         }
20
21         formulario = formulario.TipoFormularioId ==
               (int)Common.Constants.TipoFormulario.PDF
22             ? _formulariosService.AddWithFields(formulario)
23             : _formulariosService.Add(formulario);
24
25         return OkResult(formulario.Id, "Formulário criado com sucesso.");
26     }
27     catch (Exception ex) when (ex.GetBaseException() is Base64Exception)
28     {
29         _logger.LogError(ex.GetBaseException(), ex.GetBaseException().ToString(), null);
30         return Invalid("Formulário inválido");
31     }
32 }
```

Figure 38 - Add API Endpoint

Similarly, as in Figure 39, in the endpoint mapped to PUT /api/formularios/{id}/configuracoes/dadosgerais (PutFormularioDadosGerais method), when modifying an existing form, the same logic is applied to ensure that the URL format remains consistent.

```
[HttpPut("{id}/configuracoes/dadosgerais")]
[AuthorizeAny(Policy.ApiBo, Policy.ApiFormulario)]
public async Task<IActionResult> PutFormularioDadosGerais(int id,
FormularioDadosGeraisRequest data)
{
    // model validation
    if (!ModelState.IsValid)
    {
        return Invalid(ModelState);
    }

    try
    {
        var formulario = _mapper.Map<Formulario>(data);

        if (formulario.TipoFormularioId ==
            (int)Common.Constants.TipoFormulario.FORMBUILDER)
        {
            string formBuilderUrlPrefix = _configuration.GetValue<string>
                (ConfiguracoesConstants.GroupLinkFormBuilder,
                ConfiguracoesConstants.LinkFormBuilderRenderUrl);
            formulario.Ficheiro.Url = $"{formBuilderUrlPrefix}?id=
                                       {formulario.Ficheiro.Url}";
        }

        await _formulariosService.UpdateDadosGerais(id, formulario);
    }
    catch (Exception ex) when (ex.GetBaseException() is Base64Exception)
    {
        _logger.LogError(ex.GetBaseException(), ex.GetBaseException().ToString(), null);
        return Invalid("Formulário inválido");
    }

    return OkResult(id, "Formulário alterado com sucesso.");
}
```

Figure 39 - PutFormularioDadosGerais API Endpoint

This ensures that when a request is created or updated, the stored reference contains the full redirection path.

When a user attempts to edit an existing form in OnlineDesk, the system must strip out the LinkFormBuilderRenderUrl prefix so that the selected form template can be properly displayed in the dropdown selection, as seen in Figure 37.

To achieve this, the endpoint mapped to GET /api/formularios/{id}/configuracoes/dadosgerais (GetFormularioDadosGerais method)

ensures that when a Form Builder form is fetched, it processes the stored URL and extracts only the form ID as illustrated in Figure 40.

```
[HttpGet("{id}/configuracoes/dadosgerais")]
[AuthorizeAny(Policy.ApiBo, Policy.ApiFormulario)]
public IActionResult GetFormularioDadosGerais(int id)
{
    var formulario = _formulariosService.Get(id);
    // TODO [JA] [CM]: optimizar com merge do refactor
    formulario.Ficheiro = _formulariosService.GetDocumento(id);
    var response = _mapper.Map<FormularioDadosGeraisResponse>(formulario);

    if (formulario.TipoFormularioId == (int)Common.Constants.TipoFormulario.FORMBUILDER)
    {
        string formBuilderUrlPrefix = _configuration.GetValue<string>
            (ConfiguracoesConstants.GroupLinkFormBuilder,
            ConfiguracoesConstants.LinkFormBuilderRenderUrl);

        if (!string.IsNullOrEmpty(response.Formulario) &&
            response.Formulario.StartsWith(formBuilderUrlPrefix))
        {
            response.Formulario = response.Formulario.Substring
                                  (formBuilderUrlPrefix.Length + 4);
        }
    }

    return OkResult(response);
}
```

Figure 40 - GetFormularioDadosGerais API Endpoint

With these modifications, when a user creates a new request in OnlineDesk using a Form Builder template, the system correctly redirects them to the corresponding form in Form Builder, allowing them to fill it out as required.

7.2.7. Plug-in for Form Builder

To complete the integration between OnlineDesk and Form Builder, a plug-in was developed as a standalone Angular application, designed to facilitate the submission of forms directly from Form Builder to OnlineDesk. This ensures that once a user fills out a form, the submitted data is securely transmitted to OnlineDesk, along with a generated PDF attachment, maintaining a structured and auditable submission process.

Although developed externally, Form Builder provides an interface for managing and registering external plug-ins, allowing the extension of its default functionality. This capability was leveraged to introduce a custom plug-in that automates the submission of form data to OnlineDesk's API.

As shown in Figure 41, Form Builder's plug-in management interface allows for the inclusion of custom plug-ins, which ensured that OnlineDesk specific actions could be integrated into the form handling process, without requiring internal changes to the Form Builder source code. The plug-in is hosted separately and registered in Form Builder by providing its public endpoint through the platform's interface. Form Builder then loads and invokes the plug-in dynamically as part of the form submission workflow.

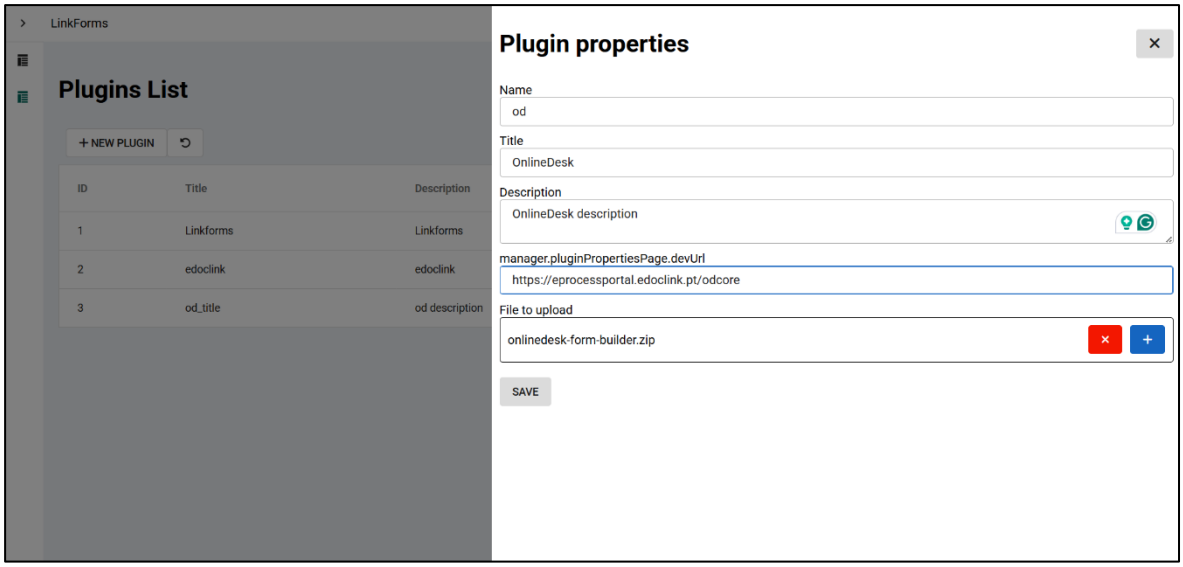


Figure 41 - Form Builder Plug-in Addition Interface

When a user submits a form within Form Builder, the plug-in is triggered. The execution begins within the execute() method, which is responsible for collecting form data, retrieving essential submission parameters, transmitting the submission to the OnlineDesk API, and finally redirect back to OnlineDesk. This process is illustrated in Figure 42.

```
1 async execute(form: ILFForm): Promise<void> { //form: ILFForm
2     this.http = this.injector!.get(HttpClient);
3
4     await new Promise<void>((resolve) => {
5       this.injector!.get(OnlinedeskConfigService).getConfig(1).then((config) => {
6           this.apiUrl = config.backendUrl;
7           resolve();
8       });
9     });
10
11    await new Promise<void>((resolve) => {
12      this.injector!.get(ActivatedRoute).queryParams.subscribe(params => {
13          this.formId = params['id'] || null;
14          this.userId = params['UtilizadorFormularioId'] || null;
15          this.token = params['token'] || null;
16          this.redirectUrl = params['RedirectUrl'] || null;
17          this.app = params['app'] || null;
18          resolve();
19      });
20    });
21
22    if (!this.apiUrl) {
23        console.error("Backend URL is not set!");
24        return;
25    }
26
27    const formData = form.getFormValue();
28    const formName = this.removeInvalidChars(form.title);
29
30    try {
31      const pdfBlob = await generatePDF(form);
32      const pdfBase64 = await this.convertBlobToBase64(pdfBlob);
33      const pdfFile = formName + '.pdf:' + pdfBase64;
34
35      const submitUrl = `${this.apiUrl}/Pedidos/${this.userId}/submeter/web`;
36      await this.submitForm(submitUrl, formData, pdfFile);
37
38
39      if (this.redirectUrl) {
40        window.location.href = this.redirectUrl;
41      }
42    } catch (error) {
43      console.error('Submission failed:', error);
44    }
45  }
46
```

Figure 42 - Execute() Method

To dynamically configure the API connection, the plug-in begins by retrieving the OnlineDesk backend URL dynamically (lines 4-9) by accessing OnlineDesk's configuration service, shown in Figure 43, through the method getConfig(), followed by extracting relevant parameters from the URL (lines 11-20).

```
1  import { Injectable } from '@angular/core';
2  import { HttpClient } from '@angular/common/http';
3
4  @Injectable({
5    providedIn: 'root'
6  })
7  export class OnlinedeskConfigService {
8    private config: any;
9    private formId: number = 1;
10
11   constructor(public http: HttpClient) { }
12
13   setFormId(formId: number) {
14     this.formId = formId;
15   }
16
17   async getConfig(pluginId: number) {
18     if (this.config) {
19       return this.config;
20     }
21     return new Promise<any>((resolve, reject) => {
22       fetch('/assets/settings.json').then(async (data: any) => {
23         const backendUrl = (await data.json())['backend']['url'];
24         this.http
25           .get(
26             backendUrl +
27               '/api/Forms/' +
28               this.formId +
29               '/PluginProperties/' +
30               pluginId
31           )
32           .subscribe((data) => {
33             this.config = data;
34             resolve(this.config);
35           });
36       });
37     });
38   }
39 }
```

Figure 43 - OnlinedeskConfigService

After that, the form data is retrieved using the getFormValue() method, and the form name undergoes a cleaning process by removing any invalid characters that may interfere with storage or transmission through the removeInvalidChars() method (lines 27-28 from Figure 42).

The removeInvalidChars() method, illustrated in Figure 44, applies a regular expression to replace restricted characters with an underscore.

```
1 private removeInvalidChars(fileName: string): string {
2   // /[\u0021-\u002f\u003a-\u0040\u005b-\u005e\u0060\u007b-\u007e]/g;
3   const invalidChars = /[<>:"/\\|?*]+/g;
4   return fileName.replace(invalidChars, "_").trim();
5 }
```

Figure 44 - removeInvalidChars() Method

Once the form data has been gathered, the next step involves generating a structured document (PDF) containing the submitted form contents. This is achieved through the generatePDF() and convertBlobToBase64() functions (lines 31-32 from Figure 42), which converts the filled form fields into a printable and easily storable format. The sanitized form name is then concatenated with the .pdf: suffix before appending the Base64-encoded PDF content (line 33 from Figure 42). This naming convention ensures correct integration with OnlineDesk's document management system.

The first function collects all input values from the form fields and structures them within a document layout, as seen in Figure 45. This ensures that the generated PDF preserves the form's structure, ensuring that the data remains accessible and properly formatted for archival or further processing.

```
export async function generatePDF(form: ILFForm, open: boolean = false) {
  return new Promise<Blob>((resolve, reject) => {
    var opts = {
      margin: [7, 7, 17, 7],
      html2canvas: { y: 0, scrollY: 0 },
      pagebreak: { avoid: '.doBreak' },
    };
    const result = generateHTML(form);
    html2pdf()
      .set(opts)
      .from(result)
      .toPdf()
      .get('pdf')
      .then((pdf: any) => {
        var totalPages = pdf.internal.getNumberOfPages();

        for (var i = 1; i <= totalPages; i++) {
          pdf.setPage(i);
          pdf.setFontSize(10);
          pdf.setTextColor(150);
          pdf.text(
            'Page ' + i + ' of ' + totalPages,
            pdf.internal.pageSize.getWidth() - 35,
            pdf.internal.pageSize.getHeight() - 10
          );
        }
        if (open) {
          var fileURL = URL.createObjectURL(pdf.output('blob'));
          window.open(fileURL);
        }

        resolve(pdf.output('blob'));
      });
  });
}
```

Figure 45 - generatePDF() Method

Once generated, the PDF is converted into Base64 format, as in Figure 46, allowing it to be uploaded as an attachment within OnlineDesk's document management system.

```
private convertBlobToBase64(blob: Blob): Promise<string> {
  return new Promise((resolve, reject) => {
    const reader = new FileReader();
    reader.readAsDataURL(blob);
    reader.onloadend = () => resolve(reader.result as string);
    reader.onerror = reject;
  });
}
```

Figure 46 - convertBlobToBae64() Method

With the PDF document ready, the plug-in executes a submission process to ensure that both the structured form data and the generated document are properly submitted and linked within OnlineDesk.

The form data submission process involves sending both the structured form fields and the generated PDF document in a single request. The plug-in extracts the filled form fields and organizes them into a JSON payload that is sent via an HTTP POST request to the /Pedidos/{userId}/submeter/web endpoint (lines 35-36 from Figure 42). This request includes both the structured form data and a reference to the uploaded PDF file. Since this endpoint was already available within the existing OnlineDesk API, the integration was designed to work with it directly, ensuring compatibility with the established submission workflow.

The method responsible for handling this submission, submitForm(), is shown in Figure 47. It takes the collected form data, formats it into a structured request, and includes the Base64-encoded PDF within the payload before submitting everything to the OnlineDesk backend.

```
1 private async submitForm(submitUrl: string, fields: any, pdfBase64: string):
  Promise<void>
2 {
3   const formattedFields = Object.entries(fields).map(([campo, valor]) => ({
4     campo,
5     valor
6   }));
7
8   const requestBody = {
9       Campos: formattedFields,
10      FicheiroFormulario: pdfBase64
11  };
12
13  const headers = {
14    'Content-Type': 'application/json',
15    'Authorization': `Bearer ${this.token}`,
16    'OD-NG-App': this.app || ''
17  };
18
19  try {
20    const response = await fetch(submitUrl, {
21      method: 'POST',
22      headers: new Headers(headers),
23      body: JSON.stringify(requestBody)
24    });
25
26  } catch (error) {
27    console.error("Form submission failed:", error);
28    throw new Error("Failed to submit the form.");
29  }
30 }
```

Figure 47 - submitForm() Method

Once the form submission is successfully completed, the final step is to redirect the user back to OnlineDesk (lines 39-41 from Figure 42). The redirection URL, extracted at the start of the process, is utilized to return the user to the appropriate OnlineDesk page.

7.2.8. Challenges and Solutions

The integration of Form Builder with OnlineDesk introduced challenges that needed to be addressed. These challenges primarily involved secure authentication, proper form retrieval and redirection, and the structured submission of form data back to OnlineDesk.

One of the main challenges was ensuring secure authentication when accessing Form Builder's API, which is protected by Keycloak. OnlineDesk needed to obtain an access token before retrieving forms while also managing token expiration efficiently. To solve this, an authentication mechanism was implemented using the OAuth 2.0 client credentials flow. OnlineDesk retrieves the necessary authentication details, such as the Keycloak base URL, realm, client ID, and client secret, from its configuration system. It then sends a request to Keycloak's token endpoint to obtain an access token, caching it along with its expiration time to reduce unnecessary requests.

Another challenge was retrieving and storing form references in a way that allowed for seamless redirection. When a user selects a form from Form Builder only its ID is retrieved in order to show the form. However, storing just the ID was not sufficient, as OnlineDesk needed to redirect users to the correct form rendering endpoint when they accessed it later. To resolve this, OnlineDesk appends the Form Builder render URL to the form ID before saving it in the database. When a user later accesses the form, OnlineDesk retrieves the stored reference, strips out the render URL when displaying the form in dropdown selections, and uses the full render URL when redirecting users.

The final challenge was handling the submission of structured form data and a PDF version of the completed form back to OnlineDesk. Since Form Builder did not have a built-in mechanism for this, a custom plug-in was developed to manage the process. When a user submits a form, the plug-in extracts all filled fields and converts the form contents into a structured PDF document to ensure traceability and compliance. The plug-in then makes an HTTP POST request to OnlineDesk's API, sending both the JSON representation of the form data and a Base64-encoded PDF file. After submission, the user is automatically redirected back to OnlineDesk, ensuring a smooth workflow.

These solutions collectively enabled a secure and automated integration between OnlineDesk and Form Builder, optimizing form-based workflows while maintaining flexibility and compliance.

7.2.9. Summary

The integration of Form Builder into OnlineDesk was implemented to provide a dynamic and customizable approach to form management without requiring modifications to OnlineDesk's internal structure. By externalizing form creation and management to Form Builder, OnlineDesk now supports flexible and customizable form handling.

This integration allows OnlineDesk to dynamically retrieve available form templates from the Form Builder repository through REST API communication while ensuring secure access to the Form Builder API via Keycloak authentication. To ensure proper form retrieval and redirection, OnlineDesk stores form references in a configurable manner. Additionally, a custom plug-in was developed within Form Builder to facilitate the structured submission of form data.

With this integration, OnlineDesk guarantees efficient and adaptable form-driven workflows. The use of externalized form management supports scalability across evolving business processes.

7.3. Integration with the Process Monitor

To improve observability and ensure accountability within workflow executions, OnlineDesk was integrated with the Process Monitor. This component of the eProcess framework collects and analyzes real-time execution events from ongoing processes, enabling detailed SLA tracking, bottleneck detection, and process compliance monitoring.

The integration is based on an event-driven architecture using Kafka as the communication layer. OnlineDesk acts as a producer, emitting structured messages that inform Process Monitor about the state and progress of tasks and workflows. These messages follow a strict format defined by the Process Monitor's message schema to ensure compatibility.

This section details the technical approach to implementing this integration, beginning with format validation, message dispatching, and the subsequent embedding of these mechanisms into OnlineDesk's workflow logic.

7.3.1. Purpose of the Integration

The integration between OnlineDesk and the Process Monitor was designed to provide real-time visibility of process execution. OnlineDesk acts as the operational layer where requests are processed and documents flow through defined workflows. However, without a monitoring mechanism, it was not possible to track SLA adherence or detect inefficiencies.

By connecting to the Process Monitor, OnlineDesk enables the capture of execution events such as task starts, completions, user assignments, and process transitions. These events are published as Kafka messages in a standardized format, providing an audit trail that feeds into dashboards and analytics tools within the eProcess platform.

This integration improves transparency, supports SLA enforcement, and enables data-driven process improvement by ensuring all relevant execution events are collected and made available in near real time. As depicted in Figure 48, the integration flow includes event triggering in OnlineDesk, dispatching messages through a provider, and consumption by the Process Monitor for analysis and monitoring purposes.

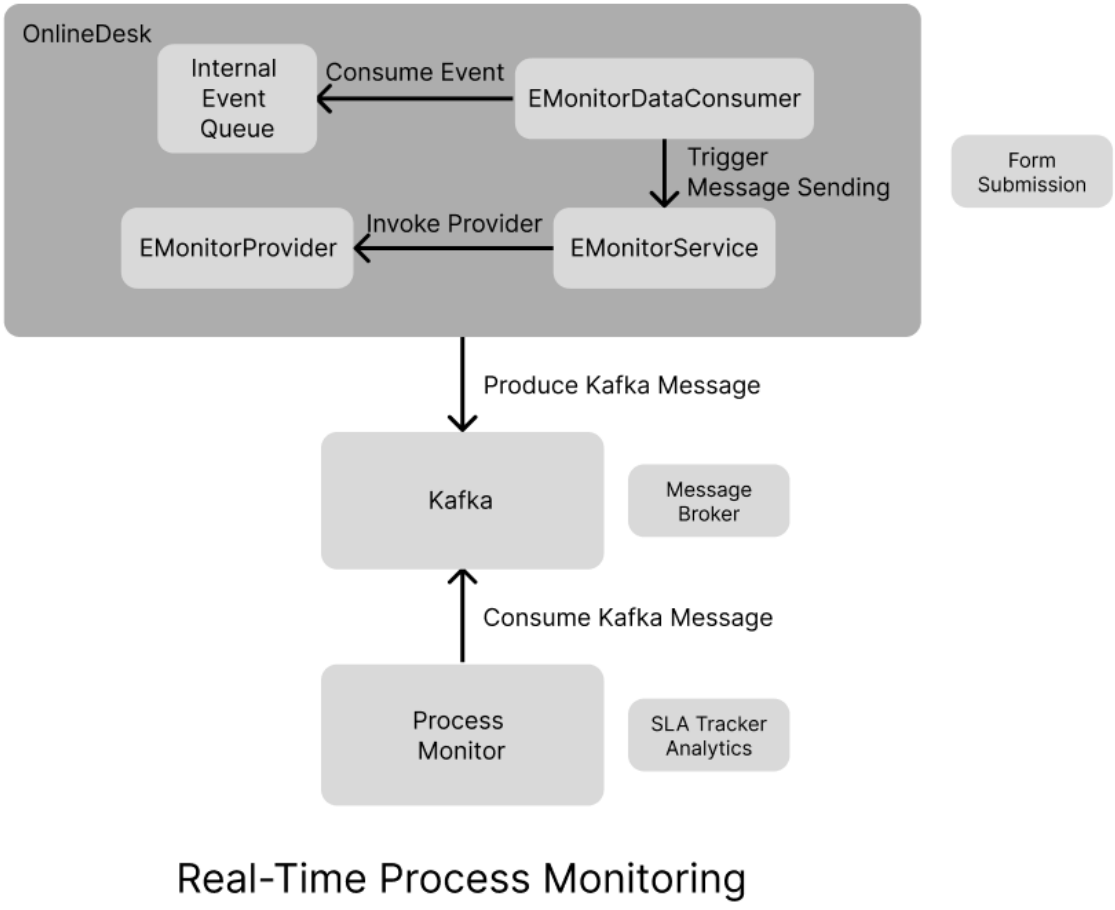


Figure 48 - Integration Flow Between OnlineDesk and Process Monitor

7.3.2. Testing Kafka Message Delivery to Process Monitor

Before implementing Kafka communication within OnlineDesk, it was essential to validate the message structure expected by the Process Monitor and ensure reliable delivery through Kafka. To achieve this, a lightweight standalone .NET test application was developed to simulate message production and isolate potential issues in message formatting, authentication, and broker connectivity.

The test application was implemented in C# using the Confluent.Kafka library. It was designed to simulate a real message dispatch to the Process Monitor topic (atlas), hosted at vwatlastest.aitec.pt:9092. The configuration required for secure communication with Kafka was defined using a ProducerConfig object, as shown in Figure 49.

```
string bootstrapServers = "vwatlastest.aitec.pt:9092";
string topic = "atlas";

var config = new ProducerConfig
{
    BootstrapServers = bootstrapServers,
    SecurityProtocol = SecurityProtocol.SaslPlaintext,
    SaslMechanism = SaslMechanism.Plain,
    SaslUsername = "eams_kafka_user1",
    SaslPassword = "Pocosi20*"
};
```

Figure 49 - Kafka ProducerConfig

This configuration defines the necessary parameters for connecting to the Kafka broker used by the Process Monitor. The BootstrapServers property specifies the address of the Kafka broker that acts as the entry point for communication. The SecurityProtocol and SaslMechanism indicate that the connection uses SASL_PLAINTEXT as the security protocol along with the PLAIN mechanism, in accordance with the requirements outlined by Process Monitor. Additionally, the SaslUsername and SaslPassword provide the credentials needed to authenticate and securely access the Kafka cluster.

Once the Kafka producer was configured, a test message adhering to the required JSON format was constructed. The message encapsulated all necessary fields, as illustrated in Figure 50, such as environmentId, environmentName, processId, versionId, type, caseId, groupName, username, taskName, processName, and isStartProcess. An outer wrapper was also applied with a "type": "EVENT" field, as described in the eProcess documentation [36].

```
var message = new
{
  "message": {
    "environmentId": 1,
    "environmentName": "od",
    "processId": "Teste OD",
    "versionId": "1.0",
    "type": "startTask",
    "caseId": "Teste Case",
    "groupName": "group-1",
    "username": "test-user",
    "taskName": "Test Task",
    "processName": "Test Process",
    "isStartProcess": true
  },
  "type": "EVENT"
}
```

Figure 50 - Kafka Message

This test setup was crucial for confirming that the Kafka producer was correctly configured and that the messages conformed to Process Monitor's expected structure. Running the test allowed to verify proper connectivity to the Kafka broker, successful message delivery to the atlas topic, and compliance with the content validation performed by the Process Monitor system. Only after this validation phase was successfully completed was the integration logic migrated into OnlineDesk's backend services, laying a reliable foundation for the full implementation of message dispatching.

7.3.3. Kafka Integration via EMonitorProvider

Following the validation of message dispatch to the Process Monitor using a standalone test application, the next step was to implement this functionality into OnlineDesk. This was achieved through the creation of a dedicated provider named EMonitorProvider.

This integration began with the definition of a new interface, IEMonitorProvider, in the OnlineDesk backend, as illustrated per Figure 51. This interface established the contract for message-sending behavior, allowing the actual implementation to be swapped or extended in the future without affecting the core application.

```
1 namespace Provider.Integration.Interfaces
2 {
3     public interface IEMonitorProvider : IProviderBase
4     {
5         Task SendMessage(string pedidoId, string username, string taskName,
                           string formConfigs, bool? startTask, bool?
                           isStartProcess);
6     }
7 }
```

Figure 51 - IEMonitorProvider Interface

The EMonitorProvider was then developed to implement this interface. Structured as a standalone provider project integrated into the OnlineDesk platform, it encapsulates the logic required to construct and send Kafka messages, adhering to the format expected by the Process Monitor. The design was focused on configurability and modularity, enabling the same implementation to be reused across different environments without code changes.

The provider dynamically loads its configuration using OnlineDesk's provider configuration system, as shown in Figure 52. These include both general settings related to Kafka communication, such as the broker host, topic name, security protocol, SASL mechanism, username, and password, and environment metadata like EnvironmentId and EnvironmentName, which are also required in the message payload.

```
public EMonitorProvider(IConfigurationOptions configurationOptions)
{
    _options = configurationOptions.Get(GetProviderName());

    if (_options != null)
    {
        _environmentId = _options.ContainsKey("EnvironmentId") ?
                         int.Parse(_options["EnvironmentId"]) : 1;
                         _environmentName =
                         _options.ContainsKey("EnvironmentName") ?
                         _options["EnvironmentName"] : null;
        _kafkaHost = _options.ContainsKey("KafkaHost") ? _options["KafkaHost"]
                        : null;
        _kafkaTopic = _options.ContainsKey("KafkaTopic") ?
                        _options["KafkaTopic"] : null;
        _kafkaSecurityProtocol = _options.ContainsKey("KafkaSecurityProtocol")
                                 ? (SecurityProtocol)
                                 Enum.Parse(typeof(SecurityProtocol),
                                 _options["KafkaSecurityProtocol"]) :
                                 SecurityProtocol.SaslPlaintext;
        _kafkaSaslMechanism = _options.ContainsKey("KafkaSaslMechanism") ?
                              (SaslMechanism)Enum.Parse(typeof(SaslMechanism),
                              _options["KafkaSaslMechanism"]) :
                              SaslMechanism.Plain;
        _kafkaSaslUsername = _options.ContainsKey("KafkaSaslUsername") ?
                             _options["KafkaSaslUsername"] : null;
        _kafkaSaslPassword = _options.ContainsKey("KafkaSaslPassword") ?
                             _options["KafkaSaslPassword"] : null;
    }
}

public IEnumerable<ProviderConfiguration> GetProviderFormConfiguration()
{
    //Configurações Gerais
    return new List<ProviderConfiguration>
    {
        new ProviderConfiguration { Key = "EnvironmentId", Type =
                                    ConfigurationType.Text },
        new ProviderConfiguration { Key = "EnvironmentName", Type =
                                    ConfigurationType.Text },
        new ProviderConfiguration { Key = "KafkaHost", Type =
                                    ConfigurationType.Text },
        new ProviderConfiguration { Key = "KafkaTopic", Type =
                                    ConfigurationType.Text },
        new ProviderConfiguration { Key = "KafkaSecurityProtocol", Type =
                                    ConfigurationType.Select, Options =
                                    Enum.GetNames(typeof(SecurityProtocol)) },
        new ProviderConfiguration { Key = "KafkaSaslMechanism", Type =
                                    ConfigurationType.Select, Options =
                                    Enum.GetNames(typeof(SaslMechanism)) },
        new ProviderConfiguration { Key = "KafkaSaslUsername", Type =
                                    ConfigurationType.Text },
        new ProviderConfiguration { Key = "KafkaSaslPassword", Type =
                                    ConfigurationType.Password }
    };
}
```

Figure 52 - Provider General Configurations

In addition to these general settings, the provider also exposes a form-level configuration model to allow customization of metadata per process, as seen in Figure 53. This includes fields like GroupName, ProcessId, VersionId, and ProcessName, which are embedded into the Kafka message to enable accurate tracking and process analytics within Process Monitor.

```
public IEnumerable<ControlBase> GetFormConfiguration()
{
    //Configurações do Formulário
    return new List<ControlInput>
    {
        new ControlInput { Key = "GroupName", Label = "Grupo", ControlType =
                           ControlInput.InputType.text },
        new ControlInput { Key = "ProcessId", Label = "Id do Processo",
                           ControlType = ControlInput.InputType.text },
        new ControlInput { Key = "VersionId", Label = "Versão", ControlType =
                           ControlInput.InputType.text },
        new ControlInput { Key = "ProcessName", Label = "Nome do Processo",
                           ControlType = ControlInput.InputType.text }
    };
}
```

Figure 53 - Provider Form Configurations

The provider constructs messages according to the format defined by the Process Monitor guidelines. Each message includes essential data such as the environment and process identifiers, user and task context, and flags like type and isStartProcess, encapsulated inside a wrapper object that includes a "type": "EVENT" field to comply with the expected Kafka schema, as in Figure 54.

```
var message = new
{
    "message": {
        "environmentId": _environmentId,
        "environmentName": _environmentName,
        "processId": formConfigData.ProcessId,
        "versionId": formConfigData.VersionId,
        "type": startTask ? "startTask" : "endTask",
        "caseId": pedidoExternoId,
        "groupName": formConfigData.GroupName,
        "username": username,
        "taskName": taskName,
        "processName": formConfigData.ProcessName,
        "isStartProcess": isStartProcess
    },
    "type": "EVENT"
}
```

Figure 54 - Kafka Message

Before the message is sent, the Kafka producer is configured using the ProducerConfig class, which is initialized with the broker's address, security protocol, SASL mechanism, and authentication credentials. A ProducerBuilder then creates the Kafka producer instance responsible for publishing messages. This is demonstrated in Figure 55.

```
var config = new ProducerConfig
{
    BootstrapServers = _kafkaHost,
    SecurityProtocol = _kafkaSecurityProtocol,
    SaslMechanism = _kafkaSaslMechanism,
    SaslUsername =  _kafkaSaslUsername, //
    SaslPassword = _kafkaSaslPassword
};

using var producer = new ProducerBuilder<Null, string>(config).Build();
```

Figure 55 - Kafka Producer

Message serialization is handled using the System.Text.Json library, which applies a JsonSerializerOptions configuration that omits optional fields when they are null to keep the payload concise. Once the message is serialized, it is dispatched to the configured Kafka topic using the asynchronous ProduceAsync() method of the Confluent.Kafka producer, completing the message delivery pipeline, as shown in Figure 56.

```
var options = new JsonSerializerOptions
{
    DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
};

string jsonMessage = JsonSerializer.Serialize(finalMessage, options);

try
{
    var deliveryResult = await producer.ProduceAsync(_kafkaTopic, new
                         Message<Null, string> { Value = jsonMessage });
    Console.WriteLine($"Message sent successfully to
                       {deliveryResult.TopicPartitionOffset}");
}
catch (Exception ex)
{
    Console.WriteLine($"Error sending message to Kafka: {ex.Message}");
}
```

Figure 56 - Message Serialization and Delivery

This provider consolidates the logic for sending task lifecycle events from OnlineDesk to Process Monitor using Kafka. It ensures consistent message formatting, secure communication, and configuration flexibility, enabling integration with the monitoring system.

7.3.4. Triggering the Kafka Provider via OnlineDesk's Monitoring Service

To integrate the newly implemented provider (EMonitorProvider) into the operational flow of OnlineDesk, a dedicated service layer was introduced. This abstraction allows the application to invoke the provider in a decoupled and reusable manner without directly depending on the provider's implementation details.

The integration starts with the definition of the IEMonitorService interface, shown in Figure 57, which declares the SendMessage() method, responsible for dispatching events to the Process Monitor through Kafka. This interface ensures flexibility for potential future enhancements or alternative implementations.

```
1 namespace Application.Core.Monitorizacao
2 {
3     public interface IEMonitorService
4     {
5         Task<bool> SendMessage(UtilizadorFormulario utilizadorFormulario,
                                 string taskName, bool? startTask, bool?
                                 isStartProcess);
6     }
7 }
```

Figure 57 - IEMonitorService Interface

The implementation of this interface, EMonitorService, contains the logic required to invoke the provider based on contextual runtime data. This service is responsible for resolving the appropriate external provider configuration, extracting the form-specific metadata, and finally send the event to the Process Monitor.

As shown in Figure 58, the service begins by searching for the form's associated provider of type IEMonitorProvider (line 8). Once the correct provider is identified, the method extracts two key pieces of information: the provider's configuration and its unique identifier (lines 14-19).

To send a message, the method SendMessage() receives an "UtilizadorFormulario" object which contains contextual information about the current request. The service then extracts relevant fields, such as the external request number and username, to populate the Kafka message fields (lines 21-22). It also handles optional flags like startTask and isStartProcess, which determine the type of lifecycle event to be sent.

Next, the provider instance is resolved using OnlineDesk's dynamic resolution mechanism (line 24), ensuring that the appropriate implementation is used with the correct configuration values.

Once all fields are resolved, the method invokes SendMessage() on the resolved provider instance (line 26), triggering the actual message dispatch to the Process Monitor.

```
1  public async Task<bool> SendMessage(UtilizadorFormulario
   utilizadorFormulario, string taskName, bool? startTask, bool? isStartProcess)
2  {
3      try
4      {
5          var formulario = await _cacheService.Formulario.FindByKeyAsync
6                                   (utilizadorFormulario.FormularioId);
7
8          var provider = formulario.FormularioProviderExterno.FirstOrDefault(p
                        => p.Tipo == nameof(IEMonitorProvider));
9          if (provider == null)
10         {
11             throw new Exception("Formulário não tem um provider do tipo E-
               Monitor");
12         }
13
14         var formConfig = formulario.FormularioProviderExterno.FirstOrDefault
15                         (p => p.Tipo ==
                           nameof(IEMonitorProvider)).Configuracao?.ToString();
16
17         var providerExternoId = formulario.FormularioProviderExterno
18                                  .FirstOrDefault
19                                  (p => p.Tipo ==
                                    nameof(IEMonitorProvider)).ProviderId;
20
21         var numeroPedidoExterno = utilizadorFormulario.Pedido?.PedidoExterno;
22         var username = utilizadorFormulario.Utilizador.Nome;
23
24         var eMonitorProvider = _eMonitorProvider.Get<IEMonitorProvider,
                                  ProviderExterno>(_configuration,
                                  providerExternoId);
25
26         await eMonitorProvider.SendMessage(numeroPedidoExterno, username,
                 taskName, formConfig, startTask, isStartProcess);
27
28         return true;
29     }
30     catch (Exception ex)
31     {
32         Console.WriteLine("Erro ao enviar mensagem para o E-Monitor");
33         return false;
34     }
35 }
```

Figure 58 - SendMessage() Method

By centralizing this functionality in a service, OnlineDesk ensures consistent and maintainable integration with Process Monitor, enabling event tracking to be triggered with minimal overhead and without duplicating Kafka-related logic throughout the codebase.

7.3.5. Frontend Integration for the EMonitorProvider

Following the backend implementation of the EMonitorProvider and its corresponding service, modifications were made to the OnlineDesk frontend to support the integration with Process Monitor. This included enabling the creation and configuration of the new provider type through the backoffice, updating provider resolution logic, and extending the form editor interface to allow provider association and customization at the form level.

To make the EMonitorProvider accessible from the OnlineDesk backoffice, the main dashboard view was updated, as shown in Figure 59.

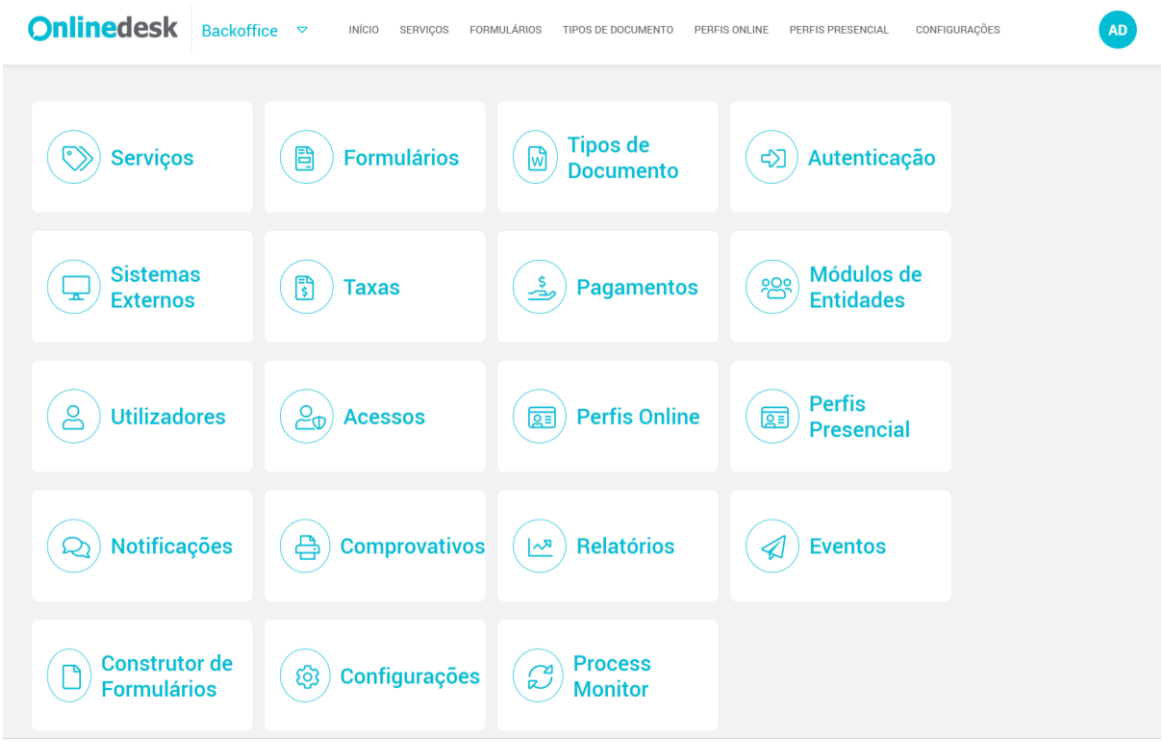


Figure 59 - Backoffice Main Dashboard

For this, a new tile labeled "Process Monitor" was added to the tiles array, linking to the route “/process-monitor”, as illustrated by Figure 60 (line 21).

```
export class BackofficeHomeComponent {
  tiles: Tile[] = [
    { title: 'Serviços', routeOption: ['../servicos'], icon: 'fa-tags',
      roles: [Roles.Servicos] },
    { title: 'Formulários', routeOption: ['../formularios'], icon: 'fa-file-
      invoice', roles: [Roles.Formularios] },
    { title: 'Tipos de Documento', routeOption: ['../tipos-documento'], icon:
      'fa-file-word', roles: [Roles.TipoDocumento] },
    { title: 'Autenticação', routeOption: ['../autenticacao'], icon: 'fa-
      sign-in-alt', roles: [Roles.Autenticacao] },
    { title: 'Sistemas Externos', routeOption: ['../sistemas-externos'],
      icon: 'fa-desktop', roles: [Roles.ServicosExternos] },
    { title: 'Taxas', routeOption: ['../modulos-taxas'], icon: 'fa-file-
      invoice-dollar', roles: [Roles.Taxas] },
    { title: 'Pagamentos', routeOption: ['../sistemas-pagamentos'], icon:
      'fa-hand-holding-usd', roles: [Roles.Pagamentos] },
    { title: 'Módulos de Entidades', routeOption: ['../modulos-entidades'],
      icon: 'fa-users', roles: [Roles.Entidades] },
    { title: 'Utilizadores', routeOption: ['../utilizadores'], icon: 'fa-
      user', roles: [Roles.Utilizadores] },
    { title: 'Acessos', routeOption: ['../utilizadores-acessos'], icon: 'fa-
      user-shield', roles: [Roles.Autenticacao] },
    { title: 'Perfis Online', routeOption: ['../perfis-online'], icon: 'fa-
      id-card', roles: [Roles.Perfis] },
    { title: 'Perfis Presencial', routeOption: ['../perfis-presencial'],
      icon: 'fa-id-card', roles: [Roles.Perfis] },
    { title: 'Notificações', routeOption: ['../sistemas-notificacoes'], icon:
      'fa-comments', roles: [] },
    { title: 'Comprovativos', routeOption: ['../documentos-provider'], icon:
      'fa-print', roles: [] },
    { title: 'Relatórios', routeOption: ['../relatorios'], icon: 'fa-chart-
      line', roles: [] },
    { title: 'Eventos', routeOption: ['../eventos'], icon: 'fa-paper-plane',
      roles: [] },
    { title: 'Construtor de Formulários', routeOption: ['../formularios-
      builder'], icon: 'fa-file', roles: [] },
    { title: 'Configurações', routeOption: ['../configuracoes'], icon: 'fa-
      cog', roles: [] },
    { title: 'Process Monitor', routeOption: ['../process-monitor'], icon:
      'fa-sync-alt', roles: [] }
  ];
```

Figure 60 - BackofficeHomeComponent Tiles

To ensure the tile navigation was properly wired, the Angular routing module was extended to register the new provider route, as seen in Figure 61. This configuration enables lazy loading of the corresponding module and specifies metadata like titles, breadcrumb, and access roles. This setup ensures that the EMonitorProvider module is accessible through the frontend navigation, with proper role-based protection and configuration mapping to the backend interface.

```
{
  path: 'process-monitor',
  loadChildren: () => import('./providers/providers.module').then((m) =>
                  m.ProvidersModule),
  canActivate: [HasRoleGuard],
  data: {
    headerTitle: 'Process Monitor',
    breadcrumbTitle: 'Process Monitor',
    roles: [Roles.Formularios],
    config: {
      entidade: 'Process Monitor',
      entidades: 'Process Monitor',
      url: 'Provider/IEMonitorProvider',
    },
  },
},
```

Figure 61 - Proccess Monitor Route

With this, the UI for configuring providers was extended to support the EMonitorProvider. The configuration interface consists of two main sections. The first one, allows the definition of the provider's display name, provider type, activation status, description, as seen in Figure 62.

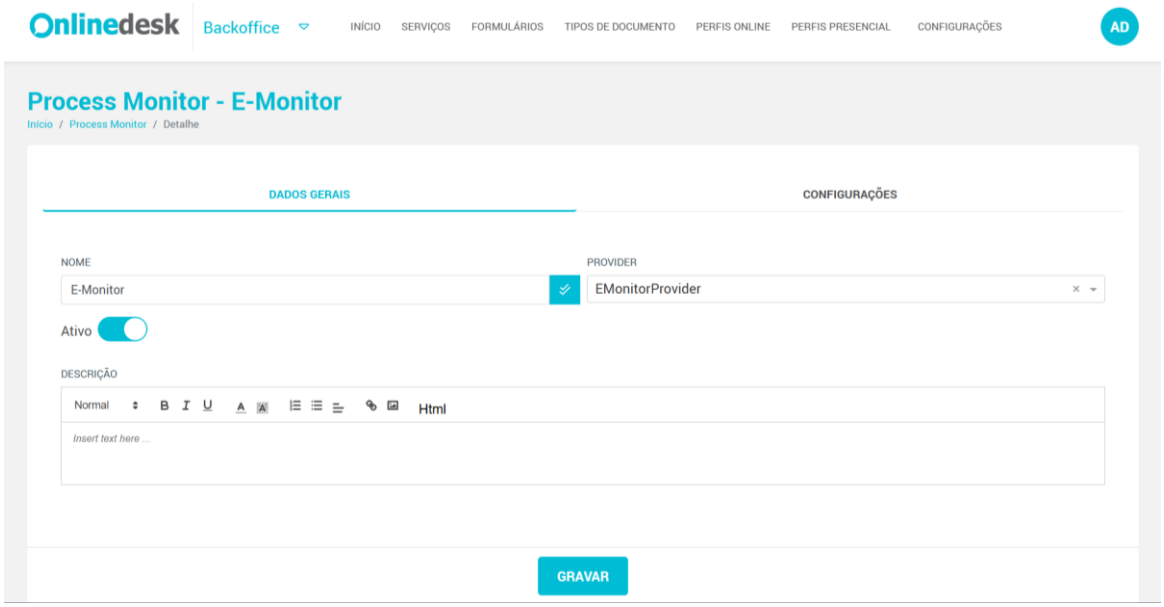


Figure 62 - Provider Definition General Data

The second section is where the Kafka-specific parameters required by the backend provider are defined, as shown in Figure 63. These parameters include the environment identifiers (EnvironmentId, EnvironmentName), Kafka broker address (KafkaHost), topic name (KafkaTopic), and authentication details (KafkaSaslMechanism, KafkaSaslUsername, and KafkaSaslPassword). Once configured, these values are saved and injected into the

provider at runtime through OnlineDesk's configuration management system, ensuring that message dispatch can be tailored per deployment environment.

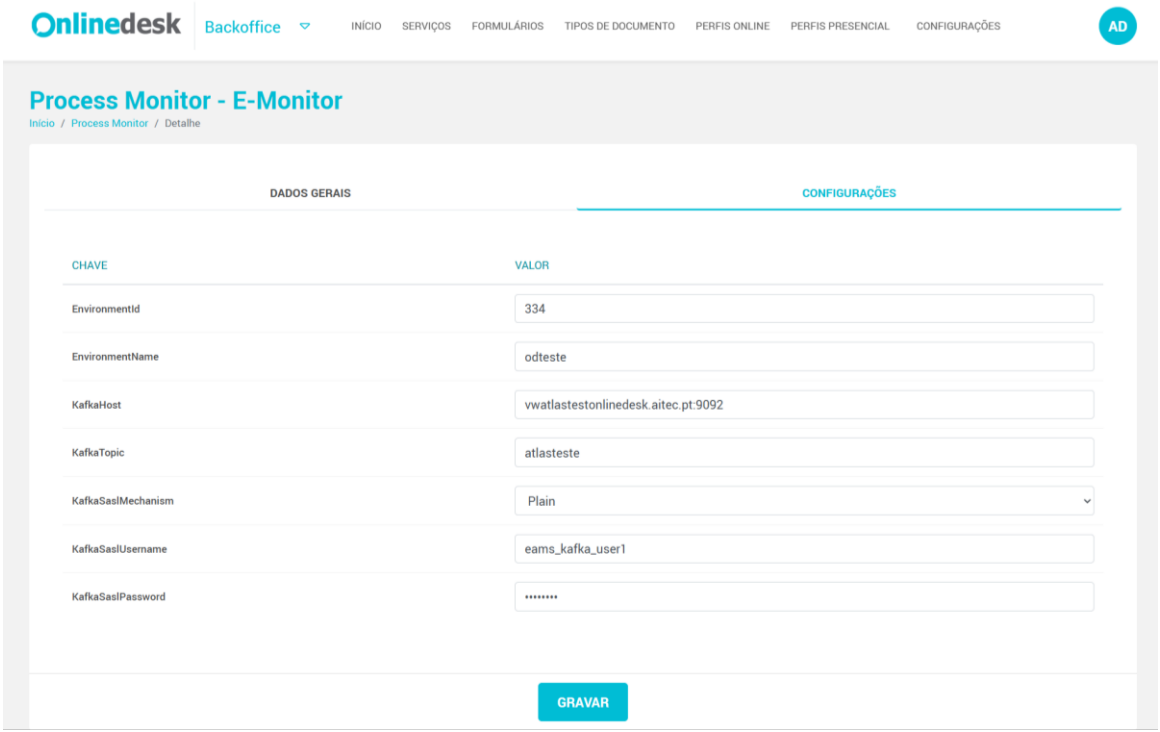


Figure 63 - Provider Definition Configurations

To enable form-level assignment of the EMonitorProvider, updates were made to the provider resolution mechanism and supporting HTTP services in the frontend. These updates ensure that the new provider appears as a selectable option within the form editor interface.

Specifically, the FormulariosProvidersService was extended to include a new method responsible for fetching EMonitorProvider instances from the backend, as illustrated in Figure 64. This method enables the frontend to retrieve available providers of this type and populate them in the list of assignable providers.

```
1 getEMonitorProvider(): Observable<Page<ProvidersList>> {
2   return this._httpClient.get<Page<ProvidersList>>
          ('provider/IEMonitorProvider').pipe(share());
3 }
```

Figure 64 - getMonitorProvider() Method

To integrate this method into the provider loading pipeline, the ProvidersResolver was also modified. It now includes a call to getEMonitorProvider() (line 9) alongside other provider-fetching methods, as shown in Figure 65.

```
export class ProvidersResolver implements Resolve<ProvidersList[]> {
  constructor(private _service: FormulariosProvidersService) {}
  resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
          Observable<ProvidersList[]> {
    return forkJoin([
      this._service.getSistemasExternos(),
      this._service.getSistemasPagamento(),
      this._service.getModulosTaxas(),
      this._service.getDocumentosProvider(),
      this._service.getEMonitorProvider(),
    ]).pipe(map((provider) => [...provider[0].items, ...provider[1].items,
                               ...provider[2].items, ...provider[3].items,
                               ...provider[4].items]));
  }
}
```

Figure 65 - ProvidersResolver

With these changes in place, if added, the EMonitorProvider becomes visible in the "Formulários" view, in the "Providers" tab of the OnlineDesk form editor UI, as depicted in Figure 66.

Figure 66 - Form Providers

Once selected, the required form-level metadata necessary for Kafka message construction is to be configured. These fields include the GroupName, the ProcessId and VersionId, and the ProcessName, as shown in Figure 67. This metadata is later injected into the Kafka message payload when a request is made using the configured form.

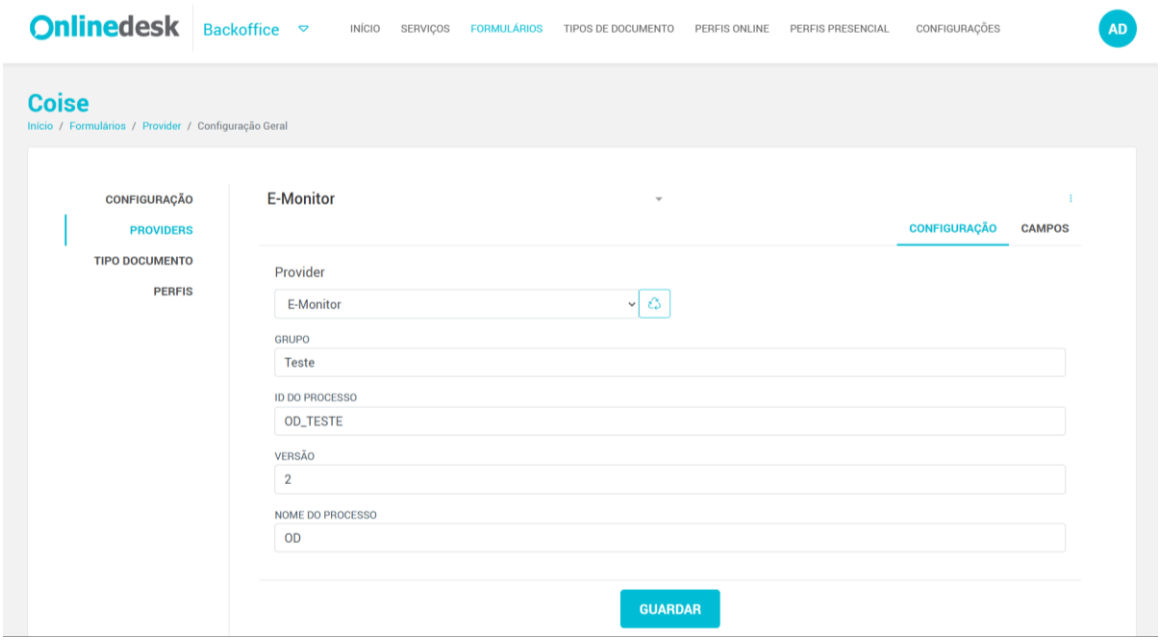


Figure 67 - Form Providers Configuration

These additions allow the integration of event tracking into form-based workflows. Specifically, when a user submits a form that has an associated EMonitorProvider, OnlineDesk emits a lifecycle event to the Process Monitor. This event includes metadata configured at the form level and enables real-time tracking.

7.3.6. Event-Based Communication with Process Monitor

To finalize the integration between OnlineDesk and Process Monitor, the system's event driven architecture was leveraged. OnlineDesk already uses a centralized event service (IEventService) to manage key operations such as sending notifications, dispatching messages to external systems, and tracking lifecycle transitions. By introducing a new domain-specific event type (SendMessageEMonitor, as shown in Figure 68 (line 30)) the system was extended to support communication with Process Monitor.

```
1  public static class EventConstants
2  {
3      public enum EventName
4      {
5          [Description("Envio Sistema Externo")]
6          EnvioSistemaExterno = 1,
7          [Description("Remover")]
8          Remover = 2,
9          [Description("Concluir Envio Sistema Externo")]
10         Concluir = 3,
11         [Description("Obter do Sistema Externo")]
12         ObterSistemaExterno = 4,
13         [Description("Pedido Pos Submissão")]
14         PostSubmit = 5,
15         [Description("Reenvio Sistema Externo")]
16         ReenvioSistemaExterno = 6,
17         [Description("Notifica Estado Pagamento")]
18         NotificaEstadoPagamento = 7,
19         [Description("Envio para o Sistema Externo Terminado")]
20         EnvioSistemaExternoTerminado = 8,
21         [Description("Envio de anexo para o Sistema Externo Terminado")]
22         EnvioAnexoTerminado = 9,
23         [Description("Envio de mensagem após submissão do pedido")]
24         EnvioMensagemPedido = 10,
25         [Description("Notificação Registada")]
26         NotificacaoRegistada = 11,
27         [Description("Envio Utilizador")]
28         EnviarUtilizador = 12,
29         [Description("Envio de Mensagem E-Monitor")]
30         SendMessageEMonitor = 13,
31     }
32 }
```

Figure 68 - EventConstants

When a new request is created and the associated form is submitted and reaches its final phase, the backend service responsible for handling form submissions (PedidosIntegrationService) triggers the publication of an SendMessageEMonitor event, as seen in Figure 69. This event contains metadata needed by Process Monitor, such as the ID of the submitted form (UtilizadorFormularioId), the task name (TaskName), a flag indicating whether the task should start (StartTask), and whether it should also mark the beginning of a process (IsStartProcess), wrapped in an EMonitorData object (line 1-7).

The event is added using the IEventService.AddAsync() method (line 9-10) and stored in the event queue, where it becomes available for processing.

```
1 var eMonitorData= new EMonitorData
2 {
3     UtilizadorFormularioId = utilizadorFormulario.Id,
4     TaskName = "Criação de Distribuição",
5     StartTask = true,
6     IsStartProcess = null
7 };
8
9 await _eventService.AddAsync(
10     EventConstants.DomainName.UtilizadorFormulario,
11     EventConstants.EventName.SendMessageEMonitor,
12     utilizadorFormulario.Id.ToString(),
13     eMonitorData);
```

Figure 69 - SendMessageEMonitor Event

On the receiving end, within the OnlineDesk backend, a consumer named EMonitorDataConsumer was created to listen to these events. As illustrated in Figure 70, once a message is consumed, it retrieves the relevant form data (line 4) and invokes the IEMonitorService.SendMessage() method (line 7), which constructs and dispatches the final message to the Process Monitor.

```
1 public async Task Consume(ConsumeContext<EMonitorData> context)
2 {
3     var eMonitorData = context.Message;
4     var utilizadorFormulario = await _utilizadorFormularioService.GetAsync
5                                     (eMonitorData.UtilizadorFormularioId);
6
7     var sendMessage = await
                        _eMonitorService.SendMessage(utilizadorFormulario,
                        eMonitorData.TaskName, eMonitorData.StartTask,
                        eMonitorData.IsStartProcess);
8
9     if (eMonitorData.QueueEventId != null)
10         _eventService.Processado((int)eMonitorData.QueueEventId);
11 }
```

Figure 70 - EMonitorDataConsumer Consume() Method

7.3.7. Challenges and Solutions

Throughout the integration with Process Monitor, several challenges emerged. One of the first decisions was whether to embed the Process Monitor logic directly within the OnlineDesk backend or to abstract it into a dedicated provider. Ultimately, a standalone provider architecture was chosen to ensure modularity and future extensibility with other systems if needed. This approach makes it easier to evolve the Process Monitor integration independently, while still fitting into the OnlineDesk provider ecosystem.

Another early challenge involved validating and understanding the entire communication pipeline with Kafka. At the time, Kafka was still a relatively unfamiliar technology within the context of OnlineDesk, so a lightweight test application was developed not only to simulate message production and consumption but also to explore how Kafka communication worked in practice. This enabled early identification of integration issues, particularly related to schema mismatches, and ensured compatibility with Process Monitor's expected message format.

Once the messaging foundation was in place, the focus shifted to configurability. Both general Kafka settings (such as broker address, topic, and authentication credentials) and form-level metadata (like group name, process ID, version, and process name) had to be made configurable through the OnlineDesk backoffice.

Finally, the last challenge was fitting the communication flow with Process Monitor into OnlineDesk's existing event-driven architecture. A new domain event type, SendMessageEMonitor, was introduced for this purpose. This required changes to the PedidosIntegrationService to emit the event when a form submission reached its final phase.

7.3.8. Summary

The integration of Process Monitor with OnlineDesk provides real-time visibility into task and process execution, enabling improved monitoring and operational transparency. By leveraging an event-driven architecture and implementing a dedicated provider, the system maintains a modular and extensible design that fits into OnlineDesk's existing provider ecosystem.

This integration supports dynamic configuration of Kafka parameters, including environment-specific metadata and form-level identifiers, ensuring adaptability across environments and use cases. Through a dedicated monitoring service and consumer logic, messages are constructed and dispatched according to Process Monitor, allowing reliable and traceable event communication.

The result is a robust and decoupled integration, enabling automatic event tracking with minimal configuration overhead, ensuring requests submitted through OnlineDesk are now transparently monitored and traceable within the Process Monitor ecosystem.

7.4.Importing Services and Forms via JSON

As part of the broader goal of improving configuration portability and interoperability across the eProcess framework, OnlineDesk was enhanced with the ability to import services and forms directly from structured JSON files. This functionality allows entity definitions to be exported from other eProcess components in a standardized format and then imported into OnlineDesk.

By defining and adhering to a common schema for services and forms, this mechanism enables consistent replication of configurations across environments and systems.

7.4.1. Purpose of the Implementation

The implementation of the JSON import functionality was driven by the need to support configuration exchange between OnlineDesk and other components within the eProcess framework in a structured and automated way.

This approach allows systems external to OnlineDesk to generate JSON files conforming to a defined schema, which can then be used to recreate services or forms in OnlineDesk without manual intervention. This enables the replication of validated configurations across multiple environments with minimal effort.

The choice of JSON as the interchange format ensures compatibility with other tools, human readability, and ease of generation or validation.

As depicted in Figure 71, this flow allows eProcess components to export standardized configuration files. These files, containing service or form definitions, are then imported into OnlineDesk through a validated and structured pipeline, enabling consistent replication across environments.

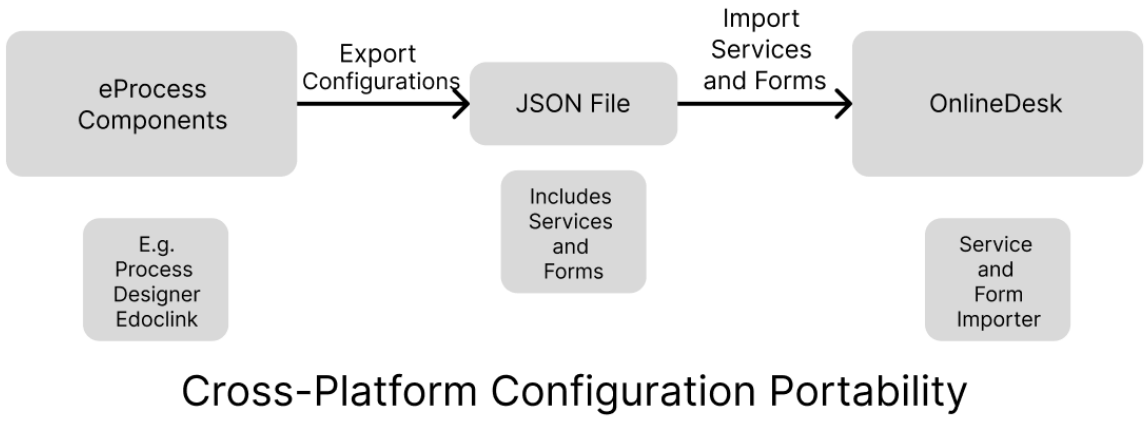


Figure 71 - JSON Import Workflow Across eProcess Components

7.4.2. Frontend Implementation of Service Import

The implementation of the JSON import functionality for services began at the frontend of the OnlineDesk backoffice. A new action button labeled "Importar Serviço" was added to the service listing interface, as in Figure 72, allowing the selection of a JSON file, with a structure similar to the one illustrated in Figure 73.

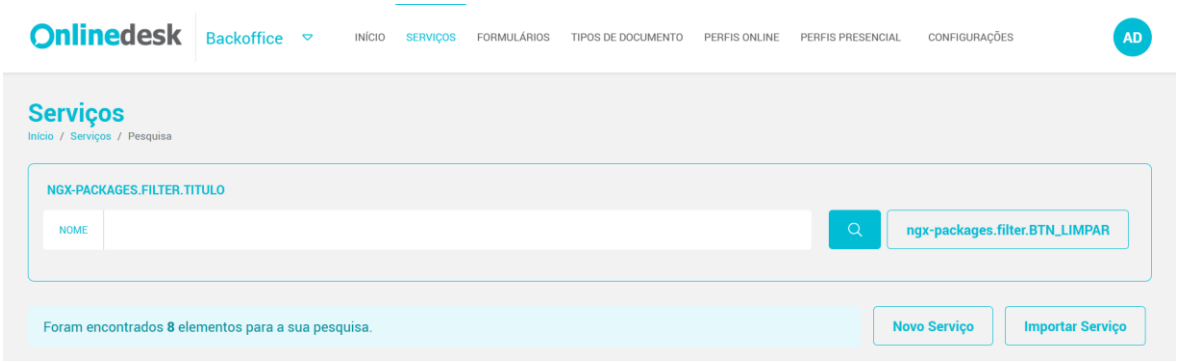


Figure 72 - "Importar Serviço" Button

```json
[
  {
    "nome": "Teste OD",
    "descricao": "Descrição do array",
    "icone": "Ícone",
    "url": "Url/Teste",
    "alias": "Alias OD",
    "servicopai": null
  },
  {
    "nome": "Filho Array",
    "descricao": "Filho do array",
    "icone": "Ícone Filho",
    "url": "Ícone Filho",
    "alias": "Alias OD Filho",
    "servicopai": "Teste OD"
  }
]
```

Figure 73 - Service Json File Structure

As shown in Figure 74, once a file is selected, the application uses a FileReader to parse its contents into JavaScript objects. The parsed data is then categorized into two groups: parent services (line 12) and child services (line 13). This distinction is based on the presence of the "servicopai" field, where services where this field is either missing or explicitly set to null are considered parents and must be created first to ensure proper reference resolution when linking children.

To preserve the service hierarchy and ensure referential integrity, a two-phase import strategy was adopted. In the first phase, all parent services (those without a "servicopai") are processed and created (line 18-20). A slight delay is introduced to allow the backend to update the service list, after which the second phase processes the child services (line 28-30). For each child, the system locates the parent by name and retrieves its identifier to establish the proper relationship. This step guarantees that each child service is linked to an existing parent, avoiding circular or broken references.

However, this solution was ultimately a naive workaround. In hindsight, a more appropriate and robust implementation would involve handling service creation as a set of asynchronous operations, awaiting each response from the backend before proceeding. This would ensure precise sequencing and eliminate reliance on arbitrary timing. The decision to rely on a delay was due to an oversight in execution flow handling, and stands out as a lesson in designing reliable client-server interactions.

```
selectFicheiro(event: Event): void {
    const ficheiro = (event.target as HTMLInputElement).files?.[0];
    if (!ficheiro) return;

    const reader = new FileReader();
    reader.onload = async (): Promise<void> => {
      try {
        const fileContents = reader.result as string;
        const parsedJson = JSON.parse(fileContents);
        const isJsonArray = Array.isArray(parsedJson) ? parsedJson :
                            [parsedJson];

        const parentServices = isJsonArray.filter(i => !i.servicopai);
        const childServices = isJsonArray.filter(i => i.servicopai);

        const existingServices = await firstValueFrom(this.data$);
        const flatServices = existingServices as unknown as Servico[];

        parentServices.forEach(parent => {
          this.handleServicoImport(parent, flatServices);
        });

        await new Promise(resolve => setTimeout(resolve, 1000));
        this.getData();

        const updatedServices = await firstValueFrom(this.data$);
        const flatUpdatedServices = updatedServices as unknown as Servico[];

        childServices.forEach(child => {
          this.handleServicoImport(child, flatUpdatedServices);
        });
      } catch (error) {
        this._toastr.error('Erro ao processar o arquivo JSON. Verifique o
                            formato.');
      }
    };

    reader.readAsText(ficheiro);
  }
```

Figure 74 - selectFicheiro() Method

The import logic is encapsulated in a dedicated method named handleServicoImport. As seen in Figure 75 and Figure 76, which includes validation and user feedback mechanisms.

```
handleServicoImport(servico: Partial<Servico> & { servicopai?: string },
existingServicos: Servico[]): void {
    servico = this.cleanServico(servico);

    if (!servico.nome) {
      this._toastr.error('JSON inválido. Certifique-se de que "nome" está
                          presente.');
      return;
    }

    if (servico.servicopai) {
      servico = {
        nome: servico.nome,
        descricao: servico.descricao,
        servicopai: servico.servicopai
      };
    }

    if (servico.servicopai) {
      const parentMatch = existingServicos.find(s => s.nome ===
                          servico.servicopai);
      if (parentMatch) {
        if (parentMatch.paiId) {
          this._toastr.error(`O serviço "${servico.servicopai}" não pode ser
                              usado como pai porque já é filho de outro
                              serviço.`);
          return;
        }

        servico.paiId = parentMatch.id;
      } else {
        this._toastr.error(`Serviço pai com nome "${servico.servicopai}" não
                            encontrado.`);
        return;
      }
    }
```

Figure 75 - handleServicoImport() Method

Before attempting to create or replace any service, the system checks for existing entries with the same name (line 32). If multiple matches are found, the import is halted, and the user is alerted to the ambiguity (line 34). In cases where a single match exists, a confirmation modal is displayed, prompting the user to approve or cancel the replacement of the existing service (line 49-51). In cases where no conflict exists, the service is imported immediately (line 59).

```
32    const matches = existingServicos.filter(s => s.nome === servico.nome);
33    if (!servico.id && matches.length > 1) {
34      this._toastr.error(`Existem vários serviços com o nome
                           "${servico.nome}". Não é possível adicionar ou
                           atualizar o serviço.`);
35      return;
36    }
37
38    const existing = matches[0];
39    const isNew = !existing;
40
41    if (!isNew) {
42      servico.id = existing!.id;
43
44      if (servico.paiId && servico.paiId === servico.id) {
45        this._toastr.error('Um serviço não pode ser pai de si próprio.');
46        return;
47      }
48
49      const modal = this.modalService.open(ConfimationModalComponent);
50      modal.componentInstance.title = 'Importar e Substituir Serviço';
51      modal.componentInstance.message = `Já existe um serviço com o nome
                                           <strong>${existing.nome}</strong>.
                                           Deseja substituí-lo?`;
52
53      modal.result.then((confirm: boolean) => {
54        if (confirm) {
55          this.uploadServico(servico);
56        }
57      });
58    } else {
59      this.uploadServico(servico);
60    }
61  }
```

Figure 76 - handleServicoImport() Method

To keep the backend implementation aligned with existing logic, the ServicosService was extended to expose a new “importar()” method. This method forwards the service object to the backend API endpoint (POST /Servicos/importar), as illustrated in Figure 77.

```
1  importar(data: Servico): Observable<Servico> {
2    return this._httpClient.post<Servico>('${this._resource}/importar', data,
                                           setProgressBar()).pipe(share());
3  }
```

Figure 77 - importar() Method

This frontend implementation provides a structured and validated entry point for service import before delegating to the backend. The next step in the implementation process involved extending the backend logic to support this functionality consistently and securely.

7.4.3. Backend Implementation of Service Import

To complement the frontend mechanism for JSON-based service import, a new API endpoint was implemented in the OnlineDesk backend to handle the persistence and validation logic. This endpoint was introduced in the ServicosController, and is responsible for interpreting the imported service data, performing business rule validation, and delegating creation or update operations to the service layer.

The new endpoint, mapped to POST /api/Servicos/importar, accepts a ServicoRequest object representing the incoming service data, as illustrated in Figure 78.

```
1 [HttpPost("importar")]
2 [Authorize(Policy.ApiBo)]
3 public IActionResult Importar([FromBody] ServicoRequest data)
```

Figure 78 - Importar Endpoint

As shown in Figure 79, the method begins by validating the model and ensuring the submitted data adheres to the constraints of the service hierarchy. The method performs several validation steps.

First, it prevents a service from referencing itself as its own parent (line 4-7). Second, if a parent Id is provided, it checks that the referenced parent exists and is not already a child of another service, thereby enforcing a strict single-level parent-child relationship (line 10-20). To handle potential duplicates, the system queries for all services with the same name (line 22-24). If multiple matches are found and no Id is provided, the operation is aborted to avoid ambiguous updates (line 25-28).

```
if (!ModelState.IsValid)
    return Invalid(ModelState);

if (data.Id.HasValue && data.PaiId.HasValue && data.Id.Value ==
data.PaiId.Value)
{
    return Invalid("Um serviço não pode ser pai de si próprio.");
}

if (data.PaiId.HasValue)
{
    var pai = _servicoService.Get(data.PaiId.Value);
    if (pai == null)
    {
        return Invalid($"O serviço pai com ID {data.PaiId.Value} não foi
                        encontrado.");
    }
    if (pai?.ServicoPaiId != null)
    {
        return Invalid("O serviço pai já é filho de outro serviço e não pode
                       ser usado como pai.");
    }
}

var allWithSameName = _servicoService
                        .GetAll(new ServicoFiltro{NomeServico = data.Nome})
                        .Where(s => s.Nome.Equals(data.Nome);
if (!data.Id.HasValue && allWithSameName.Count() > 1)
{
    return Invalid($"Existem múltiplos serviços com o nome \"{data.Nome}\".
                    Não é possível adicionar ou atualizar o serviço.");
}
```

Figure 79 - Service Validation

Finally, as seen in Figure 80, the method delegates the operation to either the Add() or Update() methods of the IServicosService, maintaining consistency with the rest of the application logic.

```
var servico = _mapper.Map<Servico>(data);
if (data.Id.HasValue && data.Id.Value > 0)
{
    var updated = _servicoService.Update(data.Id.Value, servico);
    return OkResult(_mapper.Map<ServicoResponse>(updated));
}
else
{
    var created = _servicoService.Add(servico);
    return OkResult(_mapper.Map<ServicoResponse>(created));
}
```

Figure 80 - Add or Update Service

By exposing this logic through a dedicated endpoint, the backend becomes fully compatible with the previously implemented frontend import workflow, completing the JSON-based import pipeline.

With service import fully implemented on both the frontend and backend layers, the next step involved extending similar capabilities to the import of forms, including their associations with services and external providers.

7.4.4. Frontend Implementation of Form Import

Following the successful implementation of service import, the next step involved enabling a similar mechanism for importing forms in OnlineDesk's backoffice. This functionality was introduced through an additional action in the form listing view, with a new button labeled "Importar Formulário", as illustrated in Figure 81.

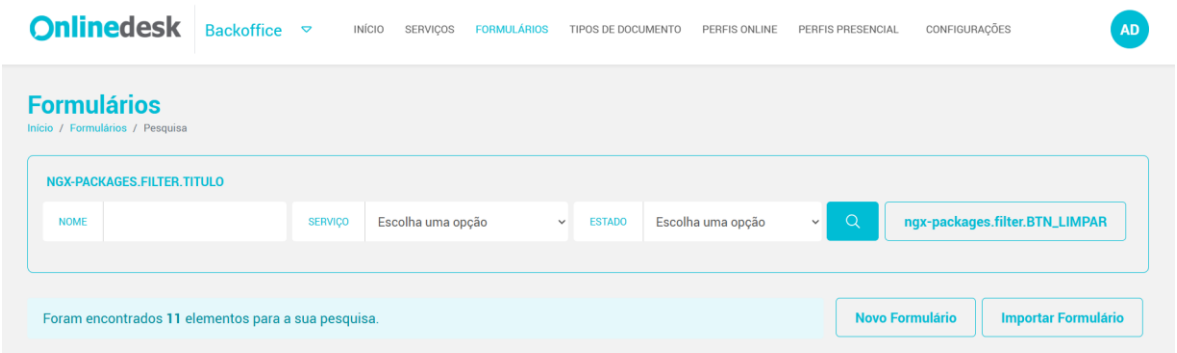


Figure 81 - "Importar Formulário" Button

Once selected, the user is prompted to upload a JSON file containing the definitions of one or more forms, with a structure similar to the one in Figure 82.

```json
[
  {
    "nome": "Web Form Example",
    "descricao": "<p>Description for web form</p>",
    "instrucoes": "<p>Please fill out all required fields.</p>",
    "tipoFormulario": "formbuilder",
    "formulario": "OD|Teste",
    "ajuda": "",
    "contacto": "contact@example.com",
    "assinaturaDigital": null,
    "preSubmit": "validateFields",
    "postSubmit": "sendToApi",
    "mapeamentoFases": "",
    "destaque": false,
    "dataInicioDestaque": null,
    "dataFimDestaque": null,
    "observacoes": "Additional notes",
    "apagado": false,
    "agendamento": false,
    "servico": "Serviço de Teste Filho",
    "nomeProvider": "Edoc",
    "classeProcesso": null,
    "livro": "TST",
    "classificacaoRegisto": null,
    "tipoDocumento": "492",
    "tipoDistribuicao": {
        "id": 31,
        "description": "",
        "acronym": "OD"
    },
    "distribuicaoPrincipal": "5 Etapas",
    "distribuicaoRelacionada": null,
    "distribuicaoSecundaria": null,
    "tramitacaoCampo": null,
    "tramitacaoValor": null,
    "assuntoProcesso": "Teste",
    "assuntoRegisto": null,
    "observacoesRegisto": null
  }
]
```

Figure 82 - Form Json File Structure

The uploaded file is parsed using a FileReader, and its contents are interpreted into JavaScript objects. Each form is validated and normalized before being submitted to the backend. This logic is triggered via the selectFicheiro() method, as seen in Figure 83, which delegates the handling of each form to the handleFormularioImport() method.

```
selectFicheiro(event: Event): void {
  const file = (event.target as HTMLInputElement).files?.[0];
  if (!file) return;

  const reader = new FileReader();
  reader.onload = async () => {
    try {
      const content = reader.result as string;
      const parsed = JSON.parse(content);
      const forms = Array.isArray(parsed) ? parsed : [parsed];

      const existingForms = await firstValueFrom(this.data$);

      for (const form of forms) {
        await this.handleFormularioImport(form, existingForms);
      }
    } catch (error) {
      this._toastr.error('Erro ao processar o arquivo. Verifique o conteúdo
                          JSON.');
    }
  };

  reader.readAsText(file);
}
```

Figure 83 - selectFicheiro() Method

The method handleFormularioImport() performs a series of validation and mapping steps to ensure that each form is valid and compatible with the system. It begins by identifying the form type through a case-insensitive key (“pfd”, "web", "htmlDesigner" or "formBuilder"), converting it to its internal tipoId representation using the TipoFormularioMap (Figure 85), as shown in Figure 84.

```
const tipoKey = (rawForm.tipoFormulario || "").toString().replace(/\s+/g,
                '').toLowerCase();
rawForm.tipoId = Formulario.TipoFormularioMap[tipoKey];
```

Figure 84 - Form Type Identification

```
export const TipoFormularioMap: Record<string, TipoFormularioEnum> = {
  pdf: TipoFormularioEnum.PDF,
  web: TipoFormularioEnum.WEB,
  htmldesigner: TipoFormularioEnum.HTML,
  formbuilder: TipoFormularioEnum.FORM_BUILDER,
};
```

Figure 85 - TipoFormularioMap

Next, the logic verifies that required fields, such as nome, tipoFormulario, and formulario, are present, as seen in Figure 86.

```
1 if (!form.nome || !form.tipoFormulario || !form.formulario) {
2   alert('JSON inválido: os campos "nome", "tipoFormulario" e "formulario" são
          obrigatórios.');
3   return;
4 }
```

Figure 86 - Required Fields Validation

Depending on the form type, it attempts to match the form name to an existing registered template. For instance, when importing an HTML Designer form, the formulario field must correspond to a known HTML template Id (line 1-10). Similarly, in FormBuilder forms, the formulario value must map to a valid template entry (line 12-21). This logic is shown in Figure 87.

```
1 if (form.tipoId === 3) {
2   const list = await firstValueFrom(this.formulariostemplates$);
3   const matched = list.find(item => item.nome === form.formulario);
4
5   if (!matched) {
6     this._toastr.error(`O formulário "${form.formulario}" não existe para
                          o tipo HTML Designer.`);
7     return;
8   }
9   form.formulario = matched.id;
10 }
11
12 if (form.tipoId === 4) {
13   const list = await firstValueFrom(this.linkformbuilder$);
14   const matched = list.find(item => item.title === form.formulario);
15
16   if (!matched) {
17     this._toastr.error(`O formulário "${form.formulario}" não existe para o t
                         ipo FormBuilder.`);
18     return;
19   }
20   form.formulario = matched.id;
21 }
```

Figure 87 - Match Form Type

If the form is associated with a specific service (via the servico field), the system attempts to locate a matching child service and assigns the corresponding servicoId. This process is illustrated in Figure 88.

```
1 if (form.servico) {
2   const allServicos = await firstValueFrom(this._formulariosService
3                       .getServicos().pipe(map((res) => res.items)));
4   const allChildServices = allServicos.map((s: Servico.List) =>
                            s.childEntities)
5                            .reduce((acc, val) => acc.concat(val), []);
6
7   const serviceMatch = allChildServices.find(s =>s.nome === form.servico);
8
9   if (!serviceMatch) {
10    this._toastr.error(`Serviço "${form.servico}" não encontrado.`);
11    return;
12  }
13  form.servicoId = serviceMatch.id;
14 }
```

Figure 88 - Match Service

Lastly, the system checks whether a form with the same name already exists (line 1). If a match is found, the user is prompted via a confirmation modal to approve the duplication (line 2-5). If the form is new or approved for import, it is submitted to the backend via the uploadFormulario() method (line 12), as shown in Figure 89.

```
1 const match = existingForms.find(f => f.nome === form.nome);
2 if (match) {
3   const modal = this.modalService.open(ConfimationModalComponent);
4   modal.componentInstance.title = 'Importar e Adicionar Formulário';
5   modal.componentInstance.message = `Já existe um formulário com o nome
                                      <strong>${match.nome}</strong>. Deseja
                                      adicionar com o mesmo nome?`;
6   const confirm = await modal.result;
7   if (!confirm) {
8     return;
9   }
10 }
11
12 this.uploadFormulario(form);
```

Figure 89 - Match Name

To keep the backend integration aligned with existing service patterns, the FormulariosService was extended to expose a new importar() method. This method receives the structured form object and forwards it to the backend API via a POST request to the /formularios/importar endpoint, as illustrated in Figure 82.

```
1 importar(formulario: Formulario.FormularioImport): Observable<number> {
2   return this._httpClient.post<number>('formularios/importar', formulario,
                                         setProgressBar());
3 }
```

Figure 90 - importar() Method

This frontend implementation provides an entry point for importing forms of various types, while ensuring proper mapping of templates, service associations, and metadata. The next step focused on implementing the corresponding backend endpoint to handle persistence, validation, and data consistency.

7.4.5. Backend Implementation of Form Import

To complement the frontend mechanism for JSON-based form import, a new API endpoint was implemented on the backend to handle the persistence, validation, and post-processing logic. This endpoint was introduced in the FormulariosController and is responsible for interpreting the imported form data, enforcing data integrity, associating services and external providers, and delegating creation operations to the service layer.

The new endpoint, mapped to POST /api/formularios/importar, accepts a FormularioImportarRequest object containing the complete metadata required to create a new form, as illustrated in Figure 91.

```
1 [HttpPost("importar")]
2 [AuthorizeAny(Policy.ApiBo, Policy.ApiFormulario)]
3 public async Task<IActionResult> Importar([FromBody] FormularioImportarRequest
                                            data)
```

Figure 91 - Importar Endpoint

As shown in Figure 92, the method begins by validating the input model (line 1-2) and ensuring required fields such as Nome, TipoId, and Formulario are present (line 4-7). It also explicitly rejects attempts to import PDF-based forms, which are unsupported within this context (line 9–12).

```
1 if (!ModelState.IsValid)
2     return Invalid(ModelState);
3
4 if (string.IsNullOrWhiteSpace(data.Nome) || data.TipoId <= 0 ||
  string.IsNullOrWhiteSpace(data.Formulario))
5 {
6     return Invalid("Os campos Nome, TipoId e Formulario são obrigatórios.");
7 }
8
9 if (data.TipoId == (int)Common.Constants.TipoFormulario.PDF)
10 {
11     return Invalid("Importação de formulários PDF não é suportada.");
12 }
```

Figure 92 - Form Validation

Then, as shown in Figure 93, the form is persisted using the Add() method of the IFormulariosService (line 1).

If a service association was defined (ServicoId), the system retrieves the specified service and validates that it is a child service (line 6-15). Only child services can be linked to forms. If valid, the association is completed using the UpdateServicosEAnexos() method (line 17).

```
1  form = _formulariosService.Add(form);
2  _context.Entry(form).State = EntityState.Detached;
3
4  if (data.ServicoId.HasValue)
5  {
6      var servico = _servicosService.Get(data.ServicoId.Value);
7      if (servico == null)
8      {
9          return Invalid($"O serviço {servico.Nome} não foi encontrado.");
10     }
11
12     if (servico.ServicoPaiId == null)
13     {
14         return Invalid($"Serviço '{servico.Nome}' não é um serviço filho e não
                       pode ser associado.");
15     }
16
17     _formulariosService.UpdateServicosEAnexos(form.Id, form,
                                          Enumerable.Empty<(int, bool)>());
18 }
```

Figure 93 - Add Form and Service

Additionally, if a provider association was defined via the NomeProvider field, the system attempts to resolve the named provider using the IProviderExternoService (line 3-7). Upon success, it constructs a structured configuration object containing the necessary metadata (line 9-28). This object is then serialized and saved using the AddProviderAsync() method (line 30-38), as seen in Figure 94.

```
if (!string.IsNullOrWhiteSpace(data.NomeProvider))
{
    var provider = await _providerExternoService.GetByName(data.NomeProvider);
    if (provider == null)
    {
        return Invalid($"Provider '{data.NomeProvider}' não foi encontrado.");
    }

    var config = new
    {
        classePorcesso = data.ClasseProcesso ?? null,
        livro = data.Livro ?? null,
        classificacaoRegisto = data.ClassificacaoRegisto ?? null,
        tipoDocumento = data.TipoDocumento ?? null,
        tipoDistribuicao = new {
            id = data.TipoDistribuicao.Id,
            description = data.TipoDistribuicao.Description,
            acronym = data.TipoDistribuicao.Acronym,
        } ?? null,
        distribuicaoPrincipal = data.DistribuicaoPrincipal ?? null,
        distribuicaoRelacionada = data.DistribuicaoRelacionada ?? null,
        distribuicaoSecundaria = data.DistribuicaoSecundaria ?? null,
        tramitacaoCampo = data.TramitacaoCampo ?? null,
        tramitacaoValor = data.TramitacaoValor ?? null,
        assuntoProcesso = data.AssuntoProcesso ?? null,
        assuntoRegisto = data.AssuntoRegisto ?? null,
        observacoesRegisto = data.ObservacoesRegisto ?? null
    };

    var providerData = new FormularioProviderExterno
    {
        FormularioId = form.Id,
        ProviderId = provider.Id,
        Configuracao = JsonConvert.SerializeObject(config),
        Tipo = provider.Tipo
    };

    var providerId = await _formulariosService.AddProviderAsync(providerData);
}
```

Figure 94 - Add Provider

By exposing this logic through a dedicated endpoint, the backend enables secure, validated, and flexible creation of forms, while supporting optional integration with both services and external providers.

With both the frontend and backend layers implemented, the form import pipeline is complete, offering a way to provision form definitions directly from structured JSON input.

7.4.6. Challenges and Solutions

Implementing the JSON-based import functionality for services and forms within OnlineDesk presented technical and architectural challenges that required solutions to maintain consistency, data integrity, and extensibility.

One of the first challenges involved designing a flexible yet controlled data structure that could support a wide variety of configurations without introducing ambiguity or unnecessary complexity. Services and forms differ significantly in complexity, where some require only basic fields, while others may include nested metadata, external provider configurations, or hierarchical relationships. To address this, a common import schema was defined for each entity in order to accommodate these variations during validation and object construction.

Another challenge was ensuring referential integrity between entities. In the case of services, the import logic needed to respect parent-child hierarchies, ensuring that parent services were created before their children could be resolved. Similarly, forms could depend on services and external providers that might not yet exist in the target environment. These associations often rely on names or identifiers that may not yet exist in the system at the time of import. To mitigate this, a two-phase import strategy was applied on the frontend, where parent entities were processed before dependent ones. On the backend, additional validations were included to ensure that associated services were valid child entries and that referenced providers were resolvable.

A further challenge involved template resolution for HTML Designer and FormBuilder forms. Because the formulario field in the import JSON could refer to a name or title rather than an internal Id, the frontend had to query available templates and perform name-based matching.

Finally, a specific challenge arose during the form import implementation on the backend due to how Entity Framework tracks entity instances within the database context. After creating a new form, the same form instance was reused during the subsequent call to associate a service to it. To resolve this, the solution involved explicitly detaching the entity from the context after creation. This ensured that subsequent operations involving the same Id could proceed without triggering tracking conflicts.

7.4.7. Summary

The implementation of JSON-based import functionality in OnlineDesk represents a significant advancement in supporting configuration portability and operational efficiency across the eProcess framework. The platform enables consistent and reusable configurations that can be exchanged between environments and integrated systems.

This capability allows external components to generate entity definitions in a unified format, which OnlineDesk can process through its validated frontend and backend workflows. The import mechanism enhances extensibility, reduces manual configuration, and promotes interoperability across the platform.

7.5.Summary

This chapter detailed the technical implementations carried out to improve OnlineDesk's architecture and interoperability within the eProcess framework. Each section addressed a specific requirement, from modernizing legacy integrations and supporting dynamic form workflows to enabling real-time monitoring and standardizing configuration import.

The migration from SOAP to REST APIs simplified external communication and improved maintainability. The integration with Form Builder introduced a flexible approach to managing form templates, while the connection to Process Monitor enabled lifecycle event tracking. Finally, the implementation of JSON-based import mechanisms allowed structured service and form configurations to be shared between environments and systems.

Together, these developments contributed to a more modular and extensible platform, aligned with ongoing integration patterns and operational needs.

8. Demonstration and Real-World Validation

This chapter presents how the results of the project were demonstrated and validated in practice. While traditional unit or integration testing was not the primary focus, all functionalities were tested iteratively during development to ensure operational correctness.

Validation was achieved through the integration of the developed features into two real-world pilot scenarios designed to showcase the capabilities of the eProcess framework. These pilots were presented in a public demonstration session held on May 8th, 2025, at the Instituto Politécnico de Leiria, in Leiria, organized under the scope of the European research initiative SIID – I&D *Empresarial – Operações Individuais* [52]. The initiative, aligned with the strategic goals of Digital Europe, promotes innovation in digital public services and supports the adoption of modern, interoperable platforms by public sector entities.

The chapter begins by detailing the selected pilot scenarios, followed by written testimonies from early adopters, and a list of public entities preparing to adopt the platform, reinforcing the system's readiness and real-world applicability.

8.1. Demonstration Use Cases

As part of the validation process supported by the European-funded eProcess project, two real-world use cases were selected to demonstrate the platform's capabilities: "Efetuar Reclamação" and "Realizar Viagem". These pilot workflows simulate practical scenarios designed to showcase the platform's applicability to real public service operations. However, it is important to clarify that only the first use case ("Efetuar Reclamação") involves direct interaction with OnlineDesk, while the second serves to highlight the broader capabilities of the eProcess framework, particularly the orchestration and archival components.

Each demonstration scenario reflects the architecture's modular nature and illustrates the platform's ability to interconnect the different components of the eProcess framework. In the case of OnlineDesk, the demonstrations validate its support for dynamic form management via Form Builder, event-based monitoring, API-driven communication with Edoclink, and configuration replication using the JSON import mechanism.

8.1.1. Efetuar Reclamação (Submit a Complaint)

This pilot use case reflects a common citizen-facing process: submitting a formal complaint to a public entity. The process begins in Atlas, the user-facing portal of the eProcess framework responsible for exposing available services and initiating request workflows through Process Designer. When the activity “Explicitar Ocorrência” (Submit Occurrence) is reached, the flow transitions to OnlineDesk.

At this point, OnlineDesk retrieves the relevant form configuration, that may have been previously imported using the JSON-based import functionality implemented during the project, ensuring consistent and reusable definitions across environments, and redirects the user to the corresponding Form Builder template.

After the user completes the form in Form Builder, a custom plugin submits the structured data and a PDF representation of the request to OnlineDesk’s backend. OnlineDesk then processes this submission by forwarding the collected information to Edoclink via the newly established REST API. In parallel, an event is emitted to the Process Monitor, marking the creation of the request and allowing for SLA tracking and process observability.

Following this submission, the remaining process is coordinated by Camunda [53], an open-source process automation platform that executes BPM workflows, which handles the internal workflow involving departments such as Secretariado, Help Desk, Reclamações, Logística, and Financeiro. Edoclink takes care of archiving and formal document management, while Process Designer continues to expose updated request statuses to the end user.

Figure 95 summarizes the integration flow and division of responsibilities among the eProcess components involved in the “Efetuar Reclamação” use case.

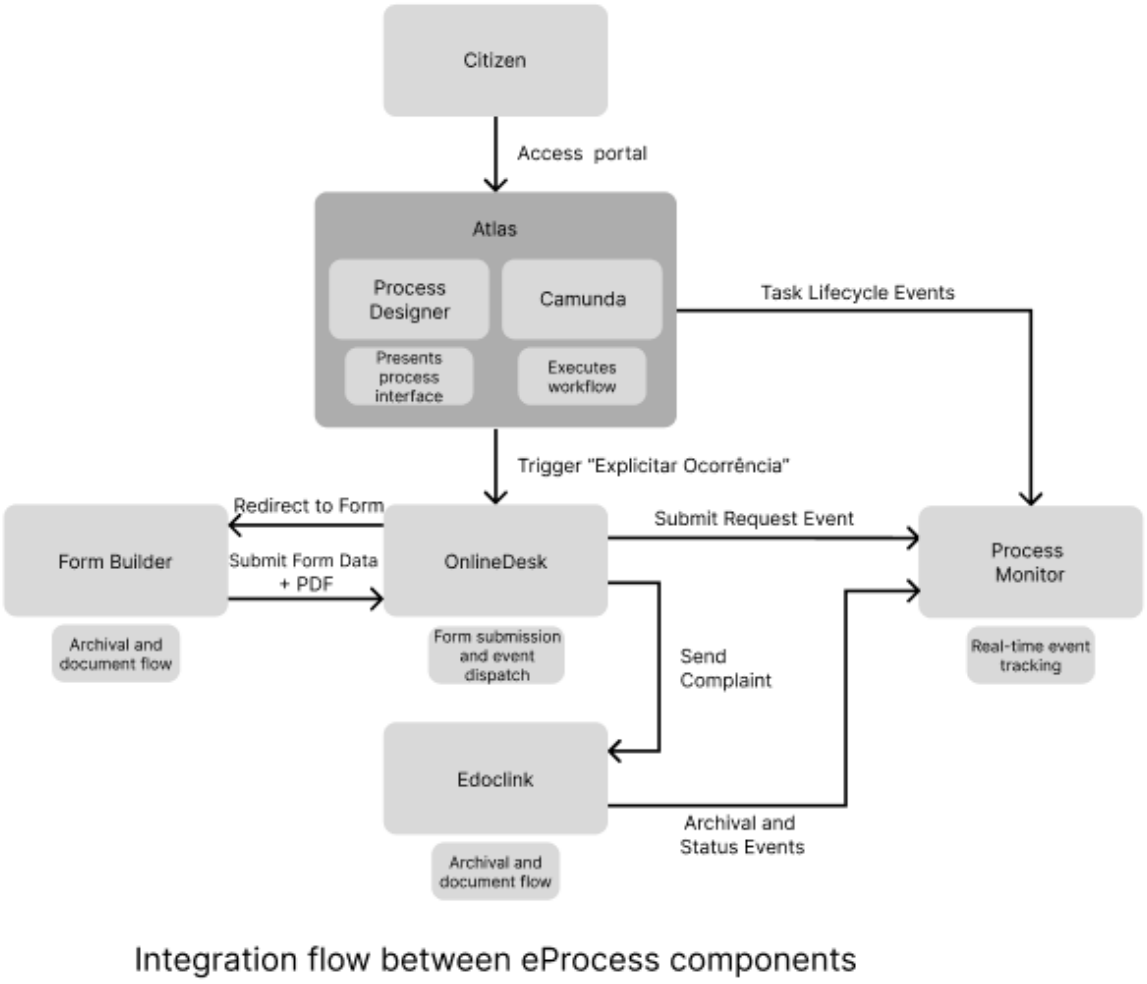


Figure 95 - Integration Flow of the "Efetuar Reclamação" Use Case

8.2.Feature Validation

Due to the iterative nature of this project and the evolving integration requirements across the eProcess framework, validation was primarily carried out through incremental functional testing throughout the implementation phase. Functionalities such as REST-based communication with Edoclink, integration with Form Builder, event emission to the Process Monitor, and the JSON-based import of services and forms were each tested in isolation as they were developed, and later verified together within complete OnlineDesk workflows.

Although the remaining components of the eProcess framework were not developed as part of this thesis, they were also validated in their respective scopes. The final pilot demonstrations served as a cross-platform validation, confirming that all systems operated correctly in a real workflow, from orchestration in Process Factory and Process Designer to data collection in OnlineDesk and archival in Edoclink.

The most comprehensive form of validation occurred during the public demonstration session held on May 8th, 2025, part of the eProcess showcase event organized under the SIID – I&D *Empresarial – Operações Individuais* initiative. The live demonstration followed the two pilot scenarios and provided end-to-end visibility into each phase of the integrated workflow, confirming the platform's operational integrity across real-world use cases [54].

This real-world demonstration reinforced confidence in the stability of the implemented features and confirmed the platform's readiness for further adoption by external entities.

8.3. Evidence of Usage

Beyond functional testing and public demonstration, the implemented features were confirmed to be in active use within the deployment of the eProcess framework. To attest to this, written statements from project supervisors and stakeholders were gathered, confirming the relevance and successful integration of the developments made during this project.

The first testimony is from Fernando Faria, director of Edoclink, and is presented in Annex 2:

"I, Fernando Faria, Director of edoclink product at DWS Unit of Link Consulting, confirm that Bernardo Lopes Ferreira contributed to the development of the OnlineDesk platform as part of his academic project taking place at IPL, with a focus on enhancing the online services component of eProcess R&D project.

His work included the implementation of several important features, namely:

- *Refactoring the existing SOAP-based integration with new edoclink version, to use RESTful API's;*
- *Enabling dynamic Form Management for online services through integration with the Form Builder module developed in eProcess;*
- *Developing an event-driven mechanism for emitting lifecycle updates to eProcess Monitor module via Kafka;*
- *Implementing a JSON-based import system to support standardized configuration of services and forms.*

These developments helped improve OnlineDesk's interoperability within the broader eProcess architecture and were successfully integrated into the platform.

Bernardo carried out his work with commitment and technical care, demonstrating autonomy and learning capacity, contributing significantly to the project's objectives."

The second testimony was provided by Miguel Santos Silva Baptista de Almeida, technical manager of OnlineDesk, technical leader and project supervisor at Link Consulting, and it's included in Annex 3:

"I, Miguel Almeida, Technical Leader at DWS, Link Consulting confirm that Bernardo Ferreira contributed to the development of the eProcess platform as part of his academic project, with a focus on enhancing the OnlineDesk component. His work included the implementation of several important features, namely:

- *Refactoring the existing SOAP-based integration with Edoclink to use RESTful APIs.*
- *Enabling dynamic form management through integration with Form Builder.*
- *Developing an event-driven mechanism for emitting lifecycle updates to Process Monitor via Kafka.*
- *Implementing a JSON-based import system to support standardized configuration of services and forms.*

These developments helped improve OnlineDesk's interoperability within the broader eProcess architecture and were successfully integrated into the production version of the platform. Bernardo carried out his work with commitment and technical care, contributing meaningfully to the project's goals."

The third testimony was given by Filipe Mendes Correia, technical leader of Edoclink and can be found in Annex 4:

"Bernardo's contribution to the development of eProcess was highly valuable, particularly in the creation of features aligned with the real requirements of the teams and the effective integration with other modules of the platform.

From early on, he showed a strong ability to understand the project context and adapt to our development practices. The work he delivered was not only functional but also

naturally integrated into the product's workflow, which facilitated its adoption and continuity by the team.

It was especially positive to see the care he took with code quality, documentation, and attention to detail - aspects that are not always evident when it comes to academic work in a business context. Bernardo demonstrated a responsible, curious, and collaborative approach, which left a positive mark on the project's evolution."

In parallel with these validations, multiple public entities are adopting the eProcess framework in production environments. As of the current date, the following entities have confirmed their migration to the platform, with deployments expected to be completed until the end of the year:

DGRSP – Direção-Geral de Reinserção e Serviços Prisionais

DGAJ – Direção-Geral da Administração da Justiça

IGSJ – Inspeção-Geral dos Serviços de Justiça

SGMJ – Secretaria-Geral do Ministério da Justiça

IGFEJ – Instituto de Gestão Financeira e Equipamentos da Justiça

DGPJ – Direção-Geral da Política de Justiça

INMLCF – Instituto Nacional de Medicina Legal e Ciências Forenses

IRN – Instituto dos Registos e Notariado

CEJ – Centro de Estudos Judiciários

Among these, both DGRSP and DGAJ have already completed their migrations and are actively operating within the new eProcess environment.

Together, this evidence demonstrates that the results of this project are not only technically valid but also contextually significant and aligned with real operational institutional needs.

9. Conclusion

This thesis presented the development and integration of several key enhancements to OnlineDesk, a core component within the eProcess framework. The work focused on improving interoperability, flexibility, and long-term maintainability through architectural and functional improvements. The implemented features included the migration from legacy SOAP-based services to REST APIs for communication with Edoclink, the integration of Form Builder to support dynamic form management, the development of a Kafka-based messaging mechanism for event emission to Process Monitor, and the introduction of a JSON-based import mechanism for reusable service and form configurations across environments.

These improvements were designed and implemented iteratively, with continuous feedback from stakeholders to ensure alignment with real operational needs. While formal unit testing was not the primary validation approach, each feature was thoroughly tested throughout development and confirmed through its integration into working workflows. The final validation occurred through a live demonstration event held under the scope of the European SIID – I&D *Empresarial – Operações Individuais* initiative, where the platform was showcased in pilot scenarios involving real-world public service use cases.

This project has reinforced OnlineDesk's role as a modular and extensible entry point in the eProcess framework. By enabling more efficient communication protocols, centralized form handling, lifecycle observability, and configuration portability, the work carried out contributes directly to the platform's scalability and future readiness.

Looking forward, several opportunities exist for further development. One potential enhancement involves expanding the JSON-based import system to support additional formats, such as CSV, and introducing export functionality to facilitate bidirectional configuration portability. This would allow entities to not only import but also back up or share service and form definitions more effectively.

Another area of interest lies in strengthening the integration with the new Edoclink REST API. This could involve expanding the set of supported operations or improving aspects such error handling and logging for better resilience and transparency.

The integration with Form Builder could also be refined by extending the custom plug-in to support dynamic pre-population of form fields based on context, conditional field logic, or even real-time validation before submission.

Lastly, expanding event traceability within Process Monitor would provide deeper operational insights. This could include capturing more granular lifecycle events, implementing custom event types, or enriching existing messages with contextual metadata to support advanced analytics and SLA tracking.

Overall, the results achieved throughout this project demonstrate how modular, standards-based design can enhance complex BPM ecosystems. The solutions implemented serve as a foundation for the continued evolution of OnlineDesk and reinforce the viability of eProcess as a modern framework for process automation.

Beyond the technical accomplishments, this project also served as a significant learning experience. It expanded my understanding of architectural patterns and strengthened my skills in full-stack development across .NET and Angular environments.

In retrospect, some design decisions could have benefited from earlier consideration or a more cautious approach. These challenges provided awareness and reinforced the importance of careful planning and validation when developing complex systems.

Throughout the process, I also gained experience translating functional requirements into technical solutions, collaborating with multidisciplinary teams, and aligning implementation choices with broader architectural goals. These aspects contributed significantly to both my technical growth and my ability to work within enterprise environments.

Bibliography

[1] Link Consulting, "Anexo Técnico: eProcess - Operações Individuais," Link Consulting, Lisbon, Portugal, 2023.

[2] P. Sousa, F. Faria, A. P. Inácio, D. Moreira, F. Correia, A. Frazao, M. Correia and D. Pereira, "LinkView Presentation - eProcess," Link Consulting, 2024.

[3] W. van der Aalst, Process Mining: Data Science in Action, Springer, 2016, pp. 3-23.

[4] Salesforce, "Salesforce Service Cloud Overview," Salesforce, 2024. [Online]. Available: https://www.salesforce.com/service/.

[5] Microsoft, "Dynamics 365 Customer Service Overview," Microsoft, 2024. [Online]. Available: https://www.microsoft.com/en-us/dynamics-365/products/customer-service.

[6] SAP, "SAP Customer Experience Overview," SAP, 2024. [Online]. Available: https://www.sap.com/products/crm.html.

[7] Oracle, "Oracle CX Service Overview," Oracle, 2024. [Online]. Available: https://www.oracle.com/cx/service/.

[8] Adobe, "Adobe Experience Manager Overview," Adobe, 2024. [Online]. Available: https://business.adobe.com/pt/products/experience-manager/adobe-experience-manager.html.

[9] Statista, "CRM Applications Vendors Market Share Worldwide," 2024. [Online]. Available: https://www.statista.com/statistics/972598/crm-applications-vendors-market-share-worldwide/#statisticContainer.

[10] CRMsearch, "CRM Software Market Share," 2024. [Online]. Available: https://crmsearch.com/crm/crm-software-market-share/.

[11] IBM, "IBM FileNet Content Manager Overview," IBM, 2024. [Online]. Available: https://www.ibm.com/br-pt/products/filenet-content-manager.

[12] OpenText, "OpenText Content Suite Overview," [Online]. Available: https://www.opentext.com/file_source/OpenText/en_US/PDF/Content-Suite-EB-Final.pdf.

[13] Link Consulting, "ONLINEDESK," Link Consulting, Lisbon, Portugal, 2021.

[14] M. V. Rosing, H. V. Scheel and A. W. Scheer, The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM, Morgan Kaufmann, 2014.

[15] M. Dumas, M. La Rosa, J. Mendling and H. A. Reijers, Fundamentals of Business Process Management, Springer, 2018.

[16] J. Dietz and H. Mulder, Enterprise Ontology, Springer, 2024, pp. 13-19.

[17] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris and D. Orchard, "Web Services Architecture," W3C Working Group Note, 2004.

[18] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures".

[19] K. e. a. Beck, "Manifesto for Agile Software Development," 2001. [Online]. Available: https://agilemanifesto.org/.

[20] J. Highsmith, Agile Project Management: Creating Innovative Products, USA: Addison Wesley Longman Publishing Co., Inc., 2009.

[21] VersionOne, "State of Agile Report," 2020.

[22] D. J. Anderson, Kanban: Successful Evolutionary Change for Your Technology Business, Blue Hole Press, 2010, p. 261.

[23] H. Kniberg, Kanban and Scrum - making the most of both, Lulu.com, 2010, p. 120.

[24] K. Leopold and S. Kaltenecker, Kanban Change Leadership: Creating a Culture of Continuous Improvement, Wiley, 2015, p. 320.

[25] A. Khoury, A. Bucknor, I. King, R. Kerstein and C. Nduka, "Use of Trello as a Project Management Tool for Collaborative Surgical Research and Audit," *British Journal of Surgery,* 2022.

[26] A. Behrens, M. Ofori, C. Noteboom and D. Bishop, "A systematic literature review: how agile is agile project," pp. 278-295, 2021.

[27] M. Shahin, M. A. Babar and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," 2017.

[28] T. Verret, "Implementing Agile Methodology: Challenges and Best Practices," 2018.

[29] A. Chatterjee and A. Prinz, "Applying Spring Security Framework with KeyCloak-Based OAuth2 to Protect Microservice Architecture APIs: A Case Study," *Sensors,* vol. 22, no. 5, p. 1703, 2022.

[30] N. Dimitrijević, N. Zdravković, M. Bogdanović and A. Mesterovic, "Advanced Security Mechanisms in the Spring Framework: JWT, OAuth, LDAP, and Keycloak," in *14th International Conference on Business Information Security (BISEC'23)*, Niš, Serbia, 2023.

[31] S. Sharad, A. Singh, Divyanshu and A. Rai, "Research Paper on Active Directory," *International Research Journal of Engineering and Technology (IRJET),* vol. 6, no. 4, p. 3579–3582, 2019.

[32] T. G. Tan, P. Szalachowski and J. Zhou, "Securing Password Authentication for Web-based Applications," 2020.

[33] Link Consulting, "Requisitos Integração com OnlineDesk - eProcess," Link Consulting, 2025.

[34] Link Consulting, "Requisitos Workflow Engine - eProcess," Link Consulting, 2025.

[35] Link Consulting, "Requisitos Integração com FormBuilder - eProcess," Link Consulting, 2025.

[36] Link Consulting, "Requisitos Process Monitor - eProcess," Link Consulting, 2025.

[37] Link Consulting, "Requisitos Process Designer - eProcess," Link Consulting, 2025.

[38] S. Chatterjee and T. Mamatha, "A comparative study on SOAP and RESTful web services," *International Research Journal of Engineering and Technology (IRJET),* vol. 7, no. 5, p. 2881–2885, 2020.

[39] [Online]. Available: https://vito-note.blogspot.com/2012/10/xml-web-service.html.

[40] P. K. Potti, S. Ahuja, K. Umapathy and Z. Prodanoff, "Comparing Performance of Web Service Interaction Styles: SOAP vs. REST," in *Proceedings of the Conference on Information Systems Applied Research*, New Orleans, LA, USA, 2012.

[41] P. Gowda and A. N. Gowda, "Best Practices in REST API Design for Enhanced Scalability and Security," *Journal of Artificial Intelligence, Machine Learning and Data Science,* vol. 2, no. 1, p. 827–830, 2024.

[42] S. P., "API 101: Securing the REST APIs," [Online]. Available: https://systemweakness.com/api-101-securing-the-rest-apis-e8b61196778f.

[43] R. Virta, "Migrating Integration from SOAP to REST: Can the Advantages of Migration Justify the Project?," University of Turku, Department of Computing, 2023.

[44] G. Geetha, M. Mittal, K. M. Prasad and J. G. Ponsam, "Interpretation and Analysis of Angular Framework," in *International Conference on Power, Energy, Control and Transmission Systems (ICPECTS)*, Chennai, India, 2022.

[45] A. Bhaskar and A. E. & Manjunath, "An Interpretation and Anatomization of Angular: A Google Web Framework," *International Research Journal of Engineering and Technology (IRJET),* vol. 7, no. 5, p. 7613–7619, 2020.

[46] K. P., "Exploring Component Architecture in Angular: A Comprehensive Guide with Examples," [Online]. Available: https://codewithpawan.medium.com/exploring-component-architecture-in-angular-a-comprehensive-guide-with-examples-b22297dffc09.

[47] M. H. Lutz and P. A. Laplante, "C# and the .NET Framework: Ready for Real Time?," *IEEE Software,* vol. 20, no. 1, p. 74–80, 2003.

[48] A. Poshtkohi, A. H. Abutalebi and S. Hessabi, "DotGrid: a .NET-based cross-platform software for desktop grids," *International Journal of Web and Grid Services,* vol. 3, no. 3, p. 313–332, 2007.

[49] Celestial Systems, "An Introduction to .NET Framework," [Online]. Available: https://celestialsys.com/blogs/an-introduction-to-net-framework/.

[50] R. Lämmel, R. Linke, E. Pek and A. Varanovich, "A Framework Profile of .NET," in *18th Working Conference on Reverse Engineering*, Limerick, Ireland, 2011.

[51] I. Khan, "Scaling .NET Core Applications to Extreme Performance," Alachisoft, 2019.

[52] Compete 2030, *SIID – I&D Empresarial – Operações Individuais,* 2024.

[53] Camunda Services GmbH, "Camunda Documentation," 2025. [Online]. Available: https://docs.camunda.io/.

[54] Link Consulting, *Evento eProcess - Leiria,* Leiria, 2025.

[55] Statista, "ERP Software Market Share by Company," 2024. [Online]. Available: https://www.statista.com/statistics/249637/erp-software-market-share-by-company/.

Annexes

Método Provider	Servico Edoc v7	Método Edoc v7	API edoc v8	Operaç	Nota
AdicionarRegisto	Cards	Add2Request	/publicapi/documents	POST	
UpdateRegisto	Cards	UpdateRequest	/publicapi/documents/{id}	PATCH	
AdicionaEntidadeAoRegisto	ServicesSimple	CardEntitiesAddRequest	/publicapi/documents/{documentId}/entities/{id}	POST	
AdicionaDocumentosAoRegistoPorN	ServicesSimple	CardDocumentsAddFullByNameRe	/publicapi/documents/{documentId}/files	POST	
SetReadyRegisto	ServicesSimple	CardSetReadyRequest	/publicapi/documents/{id}	PATCH	
GetDocumentBytes	ServicesSimple	CardDocumentVisualizationURLRe	/publicapi/files/{id}/content	GET	
AdicionaEntidadeExternaAoRegisto	ServicesSimple	CardEntityAddRequest	/publicapi/documents/{documentId}/entities/{id}	POST	
GetEdocDocumentTypes	DocumentTypes	ListRequest	/publicapi/filetypes	GET	
GetEdocDocumentTypeByInternalBo	DocumentTypes	ListAsXML2Request	/publicapi/documentTypes/{documentTypeId}/fileTypes	GET	
GetEdocCardDocumentsCount	Cards	DocumentsListRequest	/publicapi/documents/{documentId}/files	GET	
GetEdocCardDistributionsList	Cards	DistributionsListRequest	/publicapi/documents/{documentId}/flows	GET	
GetEdocCard	ServicesSimple	CardLoadRequest	/publicapi/documents/{id}	GET	
AdicionaEntidadeAoProcesso	Processes	ProcessEntityType	/publicapi/Folders/{folderId}/entities/{id}	POST	
AdicionaEntidadeExternaAoProcessc	Processes	AddEntityWithValuesRequest	/publicapi/Folders/{folderId}/entities/{id}	POST	
AddProcess	Processes	AddSRequest	/publicapi/folders	POST	
SetReadyProcess	Processes	SetReadySRequest	/publicapi/folders/{id}	PATCH	
AddCardToProcess	Processes	AddCardSRequest	/publicapi/folders/{folderId}/documents/{id}	POST	
AddFieldsToProcess	Processes	UpdateFieldsRequest	/publicapi/folders/{id}/fields	PUT	
AddSubProcessToProcess	Processes	AddSubprocessRequest	/publicapi/folders/{folderId}/subfolders/{id}	POST	Ajuda r
GetEdocDistribution	ServicesSimple	DistributionLoadRequest	/publicapi/flows/{id}	GET	
UpdateDaEtapaDaDistribuicao / Disp	ServicesSimple	DistributionStageUpdateRequest	/publicapi/flows/{flowId}/stages/{id}	PATCH	Só faze
EstadoDaDistribuicao	ServicesSimple	DistributionStatusRequest	/publicapi/flows/{id}	GET	
FaseDaDistribuicao	ServicesSimple	DistributionPhaseRequest	/publicapi/flows/{flowId}/stages	GET	não ter
ChaveCompostaDaDistribuicao	ServicesSimple	DistributionCompositeKeyRequest		GET	fazer o
ChaveDaPrimeiraDistribuicaoDoRegi	ServicesSimple	CardFirstDistributionKeyRequest	/publicapi/documents/{documentId}/flows	GET	Devolv
AdicionaDistribuicoesAoRegisto	ServicesSimple	CardDistributionsAdd1Request	/publicapi/documents/{documentId}/flows/{id}	POST	Dá erro
RemoveDistribuicaoDoRegisto	Cards	CardDistributionsDeleteRequest	/publicapi/documents/{documentId}/flows/{id}	DELETE	
AdicionaIntervenientesAEtapaDaDist	ServicesSimple	DistributionStageAddIntervenientsR	/publicapi/flows/{flowId}/stages/{id}	PATCH	Só dá p
GetEdocDistributionTemplatesList	DistributionTempla	ListRawRequest	/publicapi/flowTypes/{flowTypeId}/templates	GET	
AddFieldsToDistribution	ServicesSimple	UpdateDistributionTypeFieldsReque	/publicapi/flows/{id}/fields	PUT	Campo
GetEdocDistributionTypeFieldsList	ServicesSimple	DistributionTypeFieldsListRequest	/publicapi/flowTypes/{id}/fields	GET	
GetEdocDistributionTypeFieldsGrou	ServicesSimple	DistributionTypeFieldGroupMember	/publicapi/flowTypes/{id}/fields/{fieldId}/members	GET	
GetEdocDistributionFieldsList	ServicesSimple	DistributionFieldsListRequest	/publicapi/flows/{id}/fields	GET	Value =
GetEdocDistributionStageFieldsList	ServicesSimple	DistributionStageFieldsListRequest	/publicapi/flows/{flowId}/stages ou stages/fields?	GET	Não ter
GetEdocDistributionCurrentStageId	ServicesSimple	DistributionCurrentStagesRequest	/publicapi/flows/{id}	GET	current
AcceptDistributionStage	ServicesSimple	DistributionStageAcceptRequest	/publicapi/flows/{flowId}/stages/{id}	PATCH	Como s
UpdateEdocDistributionStageField	ServicesSimple	DistributionStageFieldUpdateReque	/publicapi/flows/{flowId}/stages/{id}/fields	PUT	Precisc
GetEdocDistributionTypesList	ServicesSimple	ListDistributionTypesRequest	/publicapi/flowTypes	GET	
GetEdocClassificationList	Classes	ListChildrenAsXmlRequest	/publicapi/classifications	GET	Quais c
GetEdocClassificationFieldsList	Classes	ClassificationFieldsListRequest	/publicapi/classifications/{id}/fields	GET	
GetEdocClassificationFieldsGroupM	Classes	ListFieldGroupMembersRequest	/publicapi/classifications/{id}/fields/{fieldId}/members	GET	Tentei
GetEdocBooksList	Books	ListAsXmlRequest	/publicapi/documentTypes	GET	
GetEdocBooksKeyByName	Books	PrimaryKeyLoadRequest	/publicapi/documentTypes/{id}	GET	
GetEdocBookFields	BookFields	ListRequest	/publicapi/documentTypes/{id}/fields	GET	
GetEdocBookFieldsGroupMemberLis	BookFields	FieldGroupMembersListRequest	/publicapi/documentTypes/{id}/fields/{fieldId}/members	GET	
GetEdocFieldXML	BookFields	GetFieldXMLRequest	/publicapi/fields/{id}	GET	Nunca
FindEntidades	Entities	EntitySearchProperty	/publicapi/entities	GET	Não se
GetEdocEntityById	Entities	GetEntityByIdAsync	/publicapi/entities/{id}	GET	
CreateEntidade	Entities	EntitiesServiceAdd4Request	/publicapi/entities	POST	
UpdateEntidade	Entities	EntitiesServiceUpdate2Request	/publicapi/entities/{id}	PUT	
AdicionaCamposDistribuicao	ServicesSimple	UpdateDistributionTypeFieldsReque	/publicapi/flows/{id}/fields	PUT	O Upd

Annex 1 - Full SOAP to REST Mapping

Declaration of participation in project

I, Fernando Faria, Director of edoclink product at DWS Unit of Link Consulting, confirm that Bernardo Lopes Ferreira contributed to the development of the OnlineDesk platform as part of his academic project taking place at IPL, with a focus on enhancing the online services component of eProcess R&D project.

His work included the implementation of several important features, namely:

- Refactoring the existing SOAP-based integration with new edoclink version, to use RESTful API's;
- Enabling dynamic Form Management for online services through integration with the Form Builder module developed in eProcess;
- Developing an event-driven mechanism for emitting lifecycle updates to eProcess Monitor module via Kafka;
- Implementing a JSON-based import system to support standardized configuration of services and forms.

These developments helped improve OnlineDesk's interoperability within the broader eProcess architecture and were successfully integrated into the platform.

Bernardo carried out his work with commitment and technical care, demonstrating autonomy and learning capacity, contributing significantly to the project's objectives.

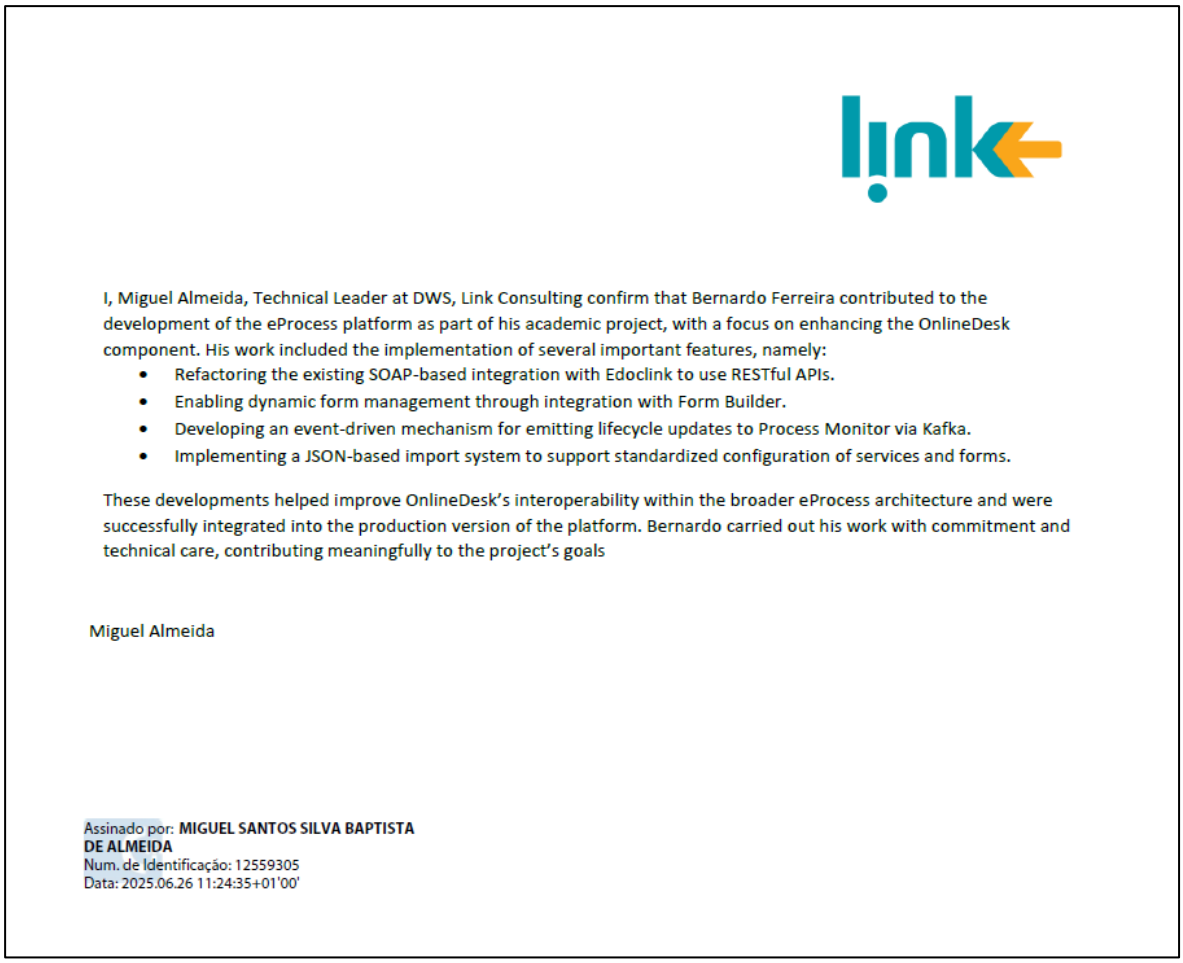
Lisbon, 23 of June 2025

FERNANDO JORGE COSTA DA SILVA FARIA

Digitally signed by FERNANDO JORGE COSTA DA SILVA FARIA
Date: 2025.06.23 16:20:07 +01'00'

(Fernando Jorge Faria)

Annex 2 - Fernando Faria's Testimony



I, Miguel Almeida, Technical Leader at DWS, Link Consulting confirm that Bernardo Ferreira contributed to the development of the eProcess platform as part of his academic project, with a focus on enhancing the OnlineDesk component. His work included the implementation of several important features, namely:

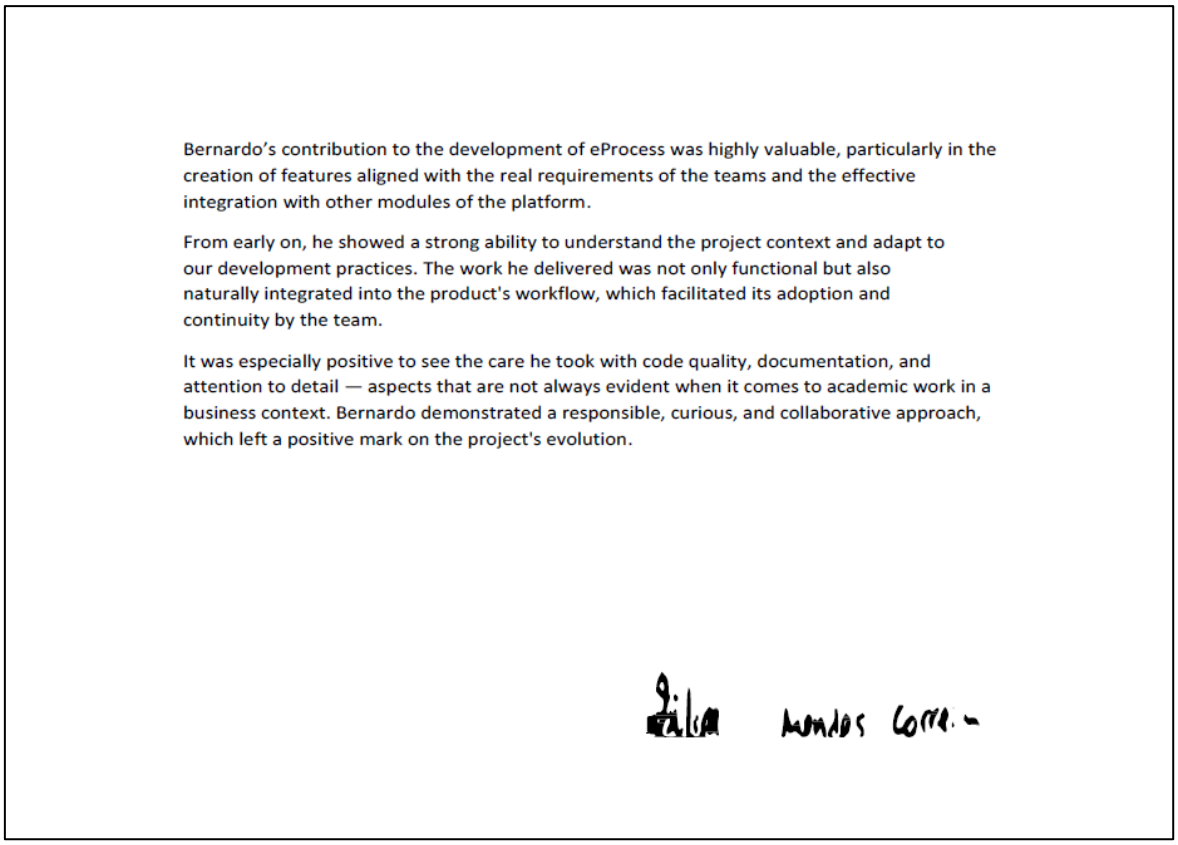
- Refactoring the existing SOAP-based integration with Edoclink to use RESTful APIs.
- Enabling dynamic form management through integration with Form Builder.
- Developing an event-driven mechanism for emitting lifecycle updates to Process Monitor via Kafka.
- Implementing a JSON-based import system to support standardized configuration of services and forms.

These developments helped improve OnlineDesk's interoperability within the broader eProcess architecture and were successfully integrated into the production version of the platform. Bernardo carried out his work with commitment and technical care, contributing meaningfully to the project's goals

Miguel Almeida

Assinado por: **MIGUEL SANTOS SILVA BAPTISTA DE ALMEIDA**
Num. de Identificação: 12559305
Data: 2025.06.26 11:24:35+01'00'

Annex 3 - Miguel Almeida's Testimony



Bernardo's contribution to the development of eProcess was highly valuable, particularly in the creation of features aligned with the real requirements of the teams and the effective integration with other modules of the platform.

From early on, he showed a strong ability to understand the project context and adapt to our development practices. The work he delivered was not only functional but also naturally integrated into the product's workflow, which facilitated its adoption and continuity by the team.

It was especially positive to see the care he took with code quality, documentation, and attention to detail — aspects that are not always evident when it comes to academic work in a business context. Bernardo demonstrated a responsible, curious, and collaborative approach, which left a positive mark on the project's evolution.

Annex 4 - Filipe Correia's Testimony